FACULTY of ENGINEERING

Process Automation Engineering Department

GRADUATION WORK

SITUATIONAL AWARENESS THROUGH THE USE OF DEEP LEARNING

by

Rahim Rahimli

Specialty: 50628 – Process Automation Engineering

Group: AM-01

Supervisor: Patrikakis Charalampos

AIGALEO – 2018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

AKNOWLEDGEMENTS

I would like to express my immense gratitude to my supervisor Assoc. Prof. **Charalampos Patrikakis** for his continuous support of my bachelor thesis research, patience, motivation, and kind attitude. Besides my adviser I would like to highlight tremendous assistance of Dr. Panagiotis Kasnesis whose contribution made this project possible. His guidance and tireless support helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Bachelor thesis study. I would like to also appreciate the care and motivation shown by Assoc. Prof. Manafaddin Namazov despite being in another country. I wish to thank my family, including parents Zamanova Nigar and Bayramov Novruz, brothers Rahimli Ibrahim, Aliyev Adalat and sister Rahimli Aysel who supported and always believed in me. I would like to also thank Hamidova Shahana for helping me to get through the challenging times, and for all the emotional support, comrade, entertainment, and caring she provided. To them I dedicate this thesis. Last but not least, I would like to express my wholehearted support to my groupmates Mammad Jabrayilov and Shamil Omarov who made this long journey unforgettable.

ABSTRACT

Situational awareness is significant for the process of the design and implementation of adaptive embedded systems. A lot of research has been conducted on the development of systems and technologies that are contextually aware of absolute position in space and time while other aspects of context have been neglected to some extent. This thesis tries to shed on light on situational awareness of machines by examining two different deep learning architectures to automatically recognize and detect environmental sounds. Acquired model exhibits higher than human accuracy level on classification of environmental auditory information which provides priceless information about a user's current context and creates possibility for a wide range of applications. The model is then successfully utilized in embedded system in the form of Raspberry Pi equipped with Movidius neural computation stick and high definition microphone to capture the sounds. This ensures mobility and provides a space for rapid deployment in various applications.

Keywords: Situational awareness, deep learning, environmental sound classification, context-aware embedded systems, convolutional neural networks for sound recognition.

INTRODUCTION

Environmental sounds can supply abundant source of information to get insights about the current situation in our near vicinity. Conversation on a mobile phone constitutes a perfect example of inferring the situation of a respondent by recognizing the background sounds which allows us to make appropriate adjustments in our voice. Widespread deployment of embedded systems combined with their reduced size and enlarged performance has led to obvious conclusion to build systems which can also be aware of situation by implicitly detecting the contextual information and adjusting their behaviour accordingly. This project aims to consider techniques for enabling machines to obtain situational awareness based on surrounding acoustic environment. Particularly, it is necessary to build an embedded system which is able to extract and classify features from pre-defined classes of sounds in the environment and detect occurrence of certain events. There are numerous vital application areas of this project including surveillance and security, response systems for ecology protection, military, context-aware mobile platforms etc. It is motivated by the idea to bring technologies closer to the human by integrating artificially intelligent systems into the field of mobile and embedded technologies. In fact, endowing machines with perception and situational awareness opens wide spectrum of possible technologies which can make human life easier and tackle challenging problems mentioned above.

This thesis is organized in accordance with the different phases of the project. First chapter gives detailed information about the situational awareness and techniques for obtaining descriptive information from time domain representation of environmental audio clips. It also explores application areas of contextual aware systems and provides brief references to necessary digital signal processing methods for pre-processing audio data including feature extraction using power spectral density, Fourier transformation of input signal and obtaining log mel spectrogram. The second chapter takes focus on thorough exploration of pattern classification techniques based on state-of-art approaches. Starting from basic notions, it gradually slides into the world of deep learning architectures and optimization techniques. The third chapter describes implementation approaches for building embedded system which is able to identify and categorize environmental sounds and thus obtain situational awareness. It also provides detailed information on architecture and design of the deep neural network. Last chapter contains detailed training and evaluation results as well as discussion of practical implementation on embedded system. Eventually there is a conclusions section which summarizes the outcomes of the project and suggests improvements for future work

Chapter 1. SITUATIONAL AWARENESS

1.1 What is situational awareness

The term "situational awareness" appeared in the military during the First World War and was related to the human perception of environmental parameters and decision making, however, nowadays, owing to the development of technology, it sounds in a new way. The first ideas about situation or context-aware systems were suggested in 1992 in the scope of the project named Active Badge system [1] however it was introduced as a term by B. Shilt *et al* [2]. This concept means the possibility of obtaining a sufficiently complete and accurate set of contextual information necessary for decision-making on the situation in real time, including the nature and characteristics of the visual and auditory surrounding, spatial location, temperature, humidity and other sensory information.

With the advent and spread of technology numerous examples of situation or context aware systems emerge in the daily life. For instance, automatic control systems for lighting installed in the entrances of residential buildings and corridors of hotels became a common phenomenon. These devices can be viewed as simple situation-aware systems. As the contextual parameters, the current state of illumination and the presence of motion nearby are taken into account [3]. The mechanism of adaptation is quite simple: if a situation is fixed, in which there is some movement in the darkness, then the lighting turns on. Then the light will be on until the person moves, and after no movement is detected for a certain period, the light will turn off again. Similarly, the light will turn off if there is light around.

Mobile smartphone is another great example of situational aware intelligent system which embraces wide spectrum of contextual information processing techniques. Equipped with broad range of sensors, modern mobile devices can adjust to numerous different circumstances and take appropriate actions such as turning screen orientation based on information from accelerometer, dim brightness to arrange optimal ergonomics, give suggestions based on location acquired from global positioning systems. Despite real-time examples, smartphones also adjust to the owner's style of typing and promote products based on his/her preferences, which is example of more general contextual awareness.

It is obvious from aforementioned examples that any form of contextual aware technology should follow a certain pattern. First step is to *analyze the situation* by making *observations* and getting sensory information from surroundings. When the current situation and its further implications are evident to decision maker it can be quite straightforward to make a conclusion on which action to accomplish [4]. After performing an action, the system should again observe and analyze whether this action had a desired effect. In other words, *decision taking* system has to measure an *error*, based on which it should make correction to further actions. This completes an iterative process known as observe, orient, decide and act loop (Figure 1.1).

According to Mica Endsley [5], situation awareness is a mental ability which she defines as the perception of the elements in the different environments, their comprehension and the projection of their status in the future. With the rise of deep learning and artificial intelligence it is possible to argue that machines can also perceive the environment, comprehend and take actions due to their ability to collect, retrieve and process vast amount of data in a blink of eye. As it was mentioned above, there are numerous methodologies to endow machines with contextual understanding of the surroundings, however this thesis focuses on the situational awareness based on environmental sounds classification.

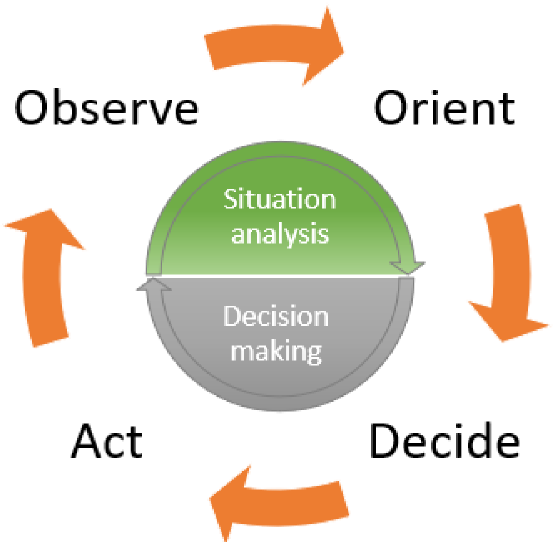


Figure 1.1 Observe, Orient, Decide Act loop

1.2 Awareness based on sound

Natural sounds give innumerable contextual cues that empower us to perceive vital facets of the surrounding world. Until recent years it was only human's prerogative to be able to classify and recognize visual or acoustic patterns because algorithms trying to accomplish similar duties were extremely inefficient. However, recent technological breakthroughs have driven a stable growth in the area of machine perception. Modern data processing units and computer systems increasingly accomplish more sophisticated tasks, occasionally transcending even human capabilities. A considerable portion of these exciting attainments has been accomplished in visual recognition, with latest spread of prosperous deep learning techniques.

In parallel, investigation on auditory classification problems has been concentrating mostly on music and speech processing. Research on environmental sounds (a manifold group of casual audio events distinct from speech and music) has retarded in applying those recent advancements. The predominant objective of the given project is to enable machines to obtain and classify features from pre-categorized classes of environmental sounds at close to human level accuracy by taking into consideration state-of-art strategies and techniques.

1.3 Motivation behind sound-based awareness. Practical applications

There are numerous vital reasons and application areas for implementation of effective situational aware system which perceives and processes auditory information around. In this context, great tribute and attention should be addressed to audio-based surveillance, safety and security systems. Although term surveillance is associated with visual observation using cameras, there are numerous applications where video recording is not possible or relevant. For example, sound aware system can efficiently and affordably detect illegal and unauthorized deforestation attempts by poachers. In contrast, it is problematic to establish visual monitoring using surveillance cameras in such environment. Sound aware systems may contribute to preventing vandalism in urban environment as it is impossible to keep track of all cameras, but intelligent context aware agent may notify of breaking glass, graffiti sounds and robbery. No less important is a natural disaster warning system which can timely inform about earthquake or flood and, thus, save innocent lives.

Another astonishing reason to measure and classify environmental sounds was given by scientist from Purdue University Bryan Pijanowski. His primary focus is to record and listen to the natural environmental sounds in the context of soundscape ecology which he refers to as a relationship between a landscape and the composition of its sound [6]. Scientist is convinced that by recording auditory information from surroundings it is possible to notice shifts in the nature caused by climate change. Numerous seemingly insignificant biological and environmental variations which can be detected by the analysis of soundscape may notify about global ecological patterns. Examples of such shifts include early emergence of insects whose lifecycles are dependent on the temperature fluctuations, while behavior of mammals and birds will stay the same. Warming regions will be quickly invaded by new species, which will potentially add their own auditory information or substitute those of native animals.

Current challenge in soundscape ecology is to figure out a way to categorize enormous amount of information that is collected from nature. A massive amount of data in the form of hours of audio recordings was collected by Pijanowski’s lab in a few years. Undeniably, it is impossible for human to listen to all this information, therefore, algorithms are needed to process it. Stuart Gage proposed an idea to take all sounds at a frequency lower than 2 kilohertz and mark it as anthropological (made by human), and then quantify acoustic energy of frequency ranges above that. Although it is accurate for many cases, certain species can produce low frequency sounds.

Another way of recognizing the sounds includes taking a frequency domain representation or, in other words, a spectrogram of the input and analyze the shape.

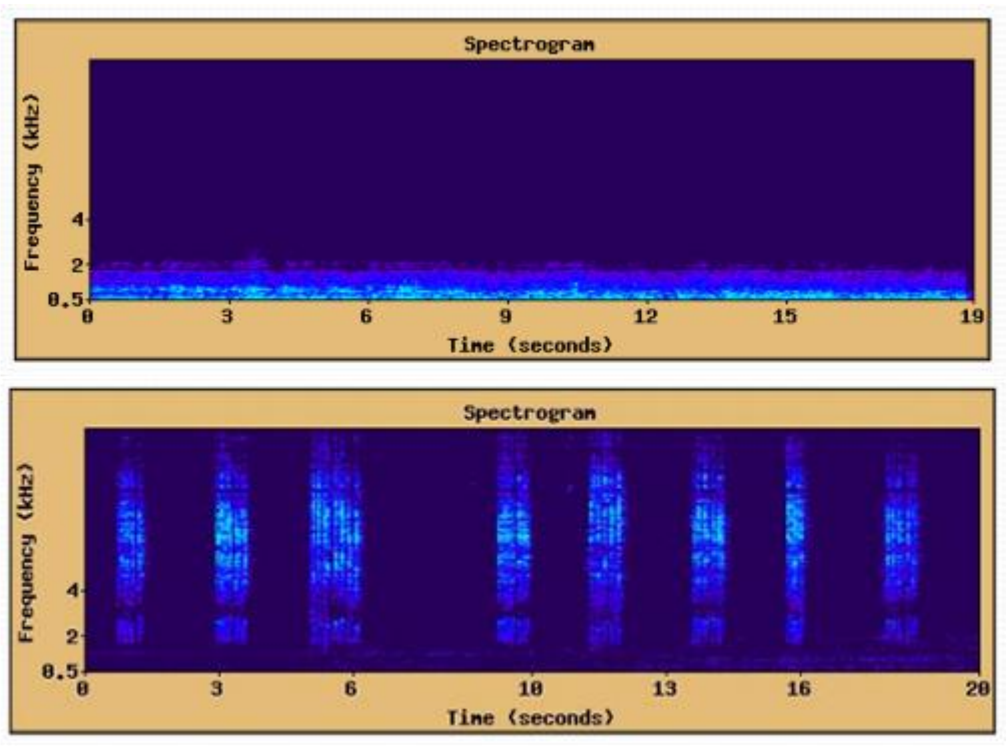


Figure.1.2 Comparison of airplane (top) and gray jay (bottom) spectrograms [41]

Biological sounds are generally sharp, fluctuating and short while anthropological sounds are monotonically steady, although both cases have exceptions. Figure.1.2 clearly illustrates solid difference between human made sounds (airplane) and biological (gray jay). It turns out that most optimal way to correctly classify and process all auditory information is to apply machine learning and train deep convolutional networks which are discussed later [7].

Recognized sounds from environment or other sources, related to kind, location and time of a certain activity can be applied for automatic delivery of dynamic and relevant data to the clients of mobile devices. In particular, users utilizing sound based wearable frameworks might be supplied with pertinent information on the base of the origins of their actual environment. Moreover, natural sounds recognition can be used to supply another crucial index into the user's context. Background sounds in such places as roads, train stations, classrooms or workspaces can provide an abundant source for final context modeling.

In order to build applications that are able to perceive and process situational information one has to apply certain number of techniques for indexing the context of a system by retrieving the sound and consecutively extracting and classifying useful features. In this thesis, it has been concentrated on a classification of five pre-determined classes of sounds in different categories by extracting a few distinctive traits. Next sections take focus on sound processing and classification techniques and discuss in detail these processes.

1.4 Sound processing and classification techniques

This section explores various methods for pre-processing sounds which are basically continuous time domain signals. The field of study dealing with representation, transformation and manipulation of signals and the information they contain is referred to as signal processing [8]. Since the sounds in computers and other microprocessors are presented in a digital form, brief information about digital signal processing techniques is provided.

1.4.1 Time and frequency representation of sounds

Sound is a longitudinal pressure wave formed of compressions and rarefactions of air molecules, in a direction parallel to that of the application of energy [9, p. 21]. Darker regions on the top of Figure 1.3 illustrate simplified image of air molecules squeezed under influence of this energy, while lighter areas stand for air zones where molecules are less densely packed. The fluctuations between compressions and rarefactions are often notionally characterized by sine wave model in time domain which is depicted in the bottom of Figure 1.3. Quantity of displacement of molecules from their original positions is measured as the *amplitude* of the signal while the duration of time for which this displacement takes place is referred to as *wavelength*. The wavelength of the signal is the reciprocal of another important parameter - *frequency* which defines number of occurrences of these displacements per unit of time.

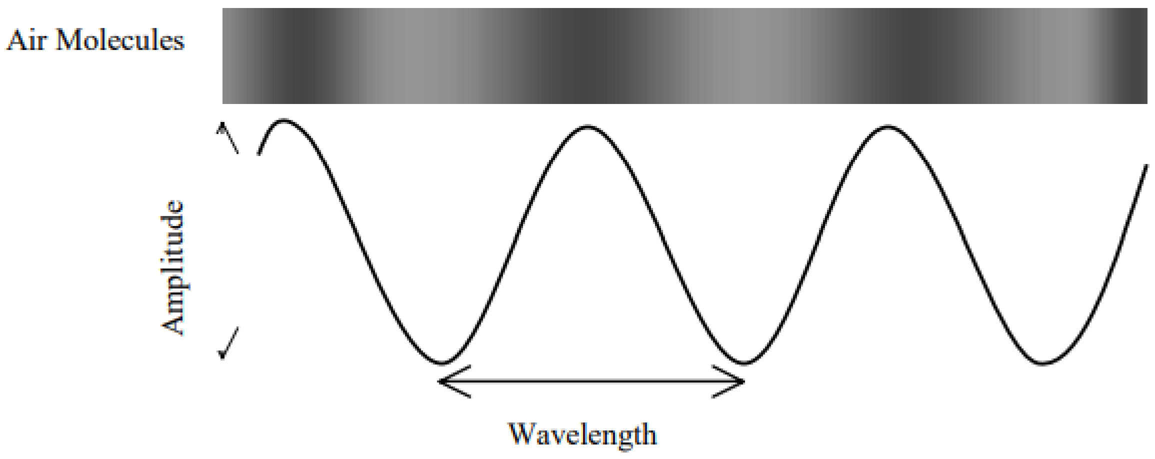


Figure 1.3 Simplified physical sound representation (top) and mathematical model in the form of sine wave (bottom) [9]

Time domain representation of the signals does not reflect much of useful information about the content of the sounds. In opposite, frequency domain qualitatively characterizes the signal and indicate which components are predominant and have the highest amplitudes. First step during analysis of any signal is to break it down from time to frequency domain and take a look at the *spectrogram*, which visually illustrates a power distribution of the different frequency components over time range. Figure.1.4 demonstrates a typical image of signal as a function of amplitude over time and its corresponding spectrogram. The brightness of a region in the spectral representation of a signal point out a presence or absence of the energy at the certain frequency. Analysis of the spectrum may assist recognition of certain patterns related to distinct classes of sounds. It is motivated by psychoacoustic experiments which were aimed to model sound perception system of human, which represented by cochlea acting as a spectrum analyzer [9].

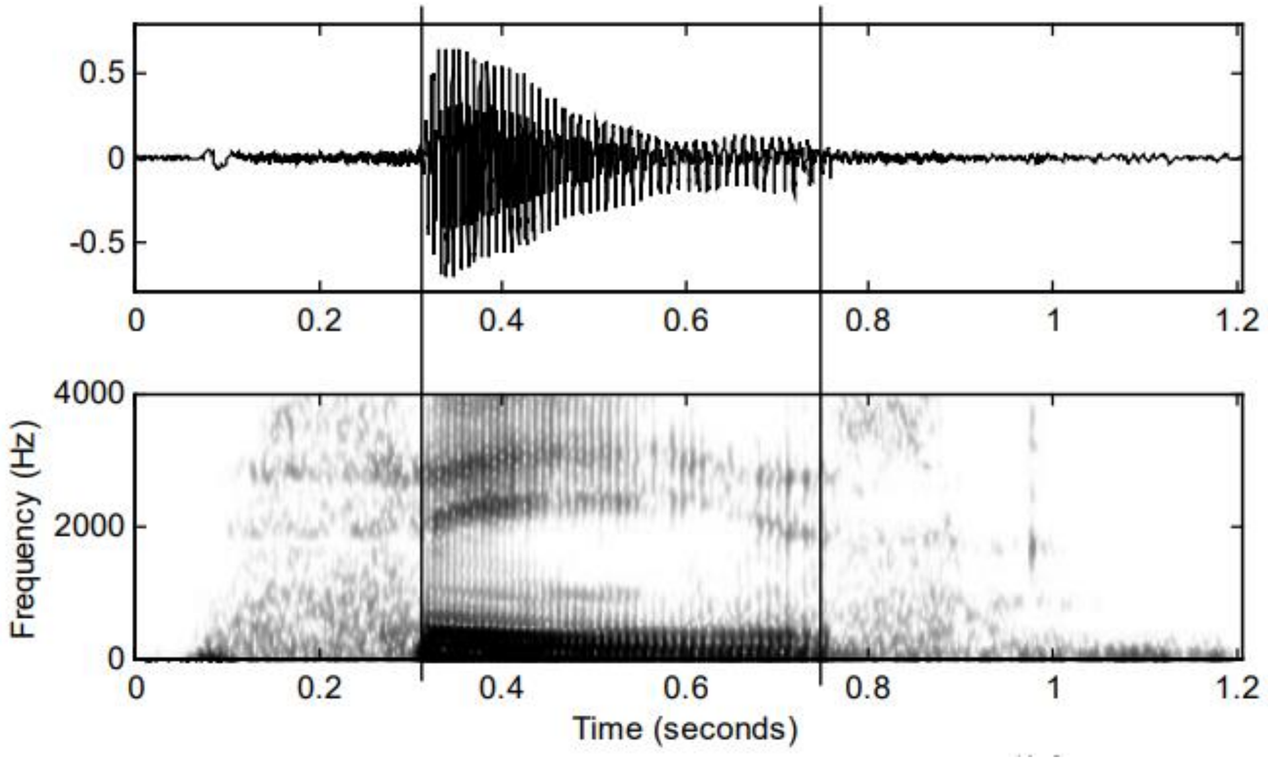


Figure.1.4 Typical representation of a signal in time domain (top) and its spectrogram(bottom) [9]

In order to convert a periodic signal from time domain into frequency representation one has to use digital signal processing technique known as Fourier transform. Equations (1-1) and (1-2) together form a Fourier representation for the discrete sequence $x[n]$ [8].

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega \tag{1-1}$$

where

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{1-2}$$

The first equation is synthesis formula, which is referred to as the inverse Fourier transform, while the second stands for Fourier transform (also referred to as the discrete-time Fourier transform (DTFT)) which expresses a way to compute arbitrary signal $X(e^{j\omega})$ from sequence $x[n]$, i.e. for analyzing the sequence $x[n]$ to determine how much of each frequency component is required to synthesize $x[n]$ using (1-1) [8]. There are numerous implementations of this technique, however, different variations of the algorithm known as fast Fourier transform found the most widespread utilization. One of the most common realization of this algorithm was introduced by J.W. Tukey and J.W. Cooley [10] and applies recursive discrete Fourier transform to an arbitrary input signal. Pseudo implementation of this algorithm can be found in Appendix 1.

1.4.2 Feature extraction. Mel Frequency Cepstral representation

As mentioned earlier, in order to properly classify sound signals, in the first place one has to extract valuable descriptive information from their time domain representation and place it in a compact set of feature vectors [11]. This process is known as *feature extraction* and is widely spread in the field of human speech recognition. Computing Fourier transform of short periodic signals by using digital signal processing techniques is reasonable and justified for feature extraction. However, this transformation requires knowledge of the signal for infinite time and inapplicable in many cases due to the fact that sounds usually consist of aperiodic segments [9]. For these reasons, short-time analysis techniques are used in order to decompose sound signal into the range of short pieces known as analysis frames.

One the of most successful forms of feature extraction from short-time signals is *Mel-Frequency cepstral (MFC)* representation. It is defined as the real *cepstrum* of a windowed short-time signal derived from the FFT of that signal on a non-linear mel scale [9]. Cepstrum here stands for the outcome of the inverse Fourier transform of the logarithm of a signal’s approximated spectrum. Cepstrum analysis is needed for extracting a spectral envelope of a signal which is extremely beneficial as source of features for classification. Spectral envelope carries the identity of a sound and is defined as a smooth curve (Figure 1.5) connecting peaks or dominant frequency components, which are referred to as formants. Appendix 1 provides brief steps to undertake cepstrum analysis [12].

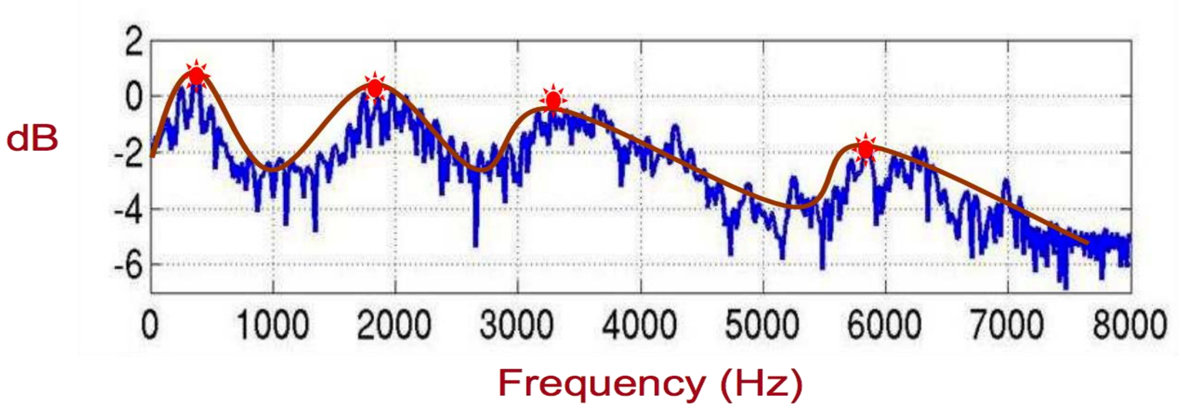


Figure 1.5.Spectral envelope of a signal [12]

In contrast with real cepstrum, MFC is defined on non-linear frequency scale and bands are equally positioned on the Mel scale, which resembles human auditory system more accurately than frequency bands in the normal spectrum. This is explained by the fact that human ears act like a filter which focuses only on specific frequency components which non-uniformly situated on the frequency axis. The main concept is to have higher number of filters in low frequency region and less in the high frequencies zone. Having discrete Fourier transform in the form of (1-3) one can define filterbank with M mel filters (m = 1,2… M), where m-th filter is defined by (1-4).

$$X_a[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N} \tag{1-3}$$

$$H_m[k] = \begin{cases} 0 & k < f[m-1] \\ \dfrac{2(k-f[m-1])}{(f[m+1]-f[m-1])(f[m]-f[m-1])} & f[m-1] \le k \le f[m] \\ \dfrac{2(f[m+1]-k)}{(f[m+1]-f[m-1])(f[m+1]-f[m])} & f[m] \le k \le f[m+1] \\ 0 & k > f[m+1] \end{cases} \tag{1-4}$$

Figure 1.6 clearly demonstrates non-linear spacing of triangular mel filters calculated using (1-4). It is noteworthy to mention that filters are tightly packed in low frequency regions, while in the area of the high frequencies they are relatively sparse. Translating spectrum into mel scale using these filters and applying consecutive cepstral analysis produces Mel-Frequency Cepstral Coefficients (MFCC) which immensely contribute to the performance of pattern classifiers.

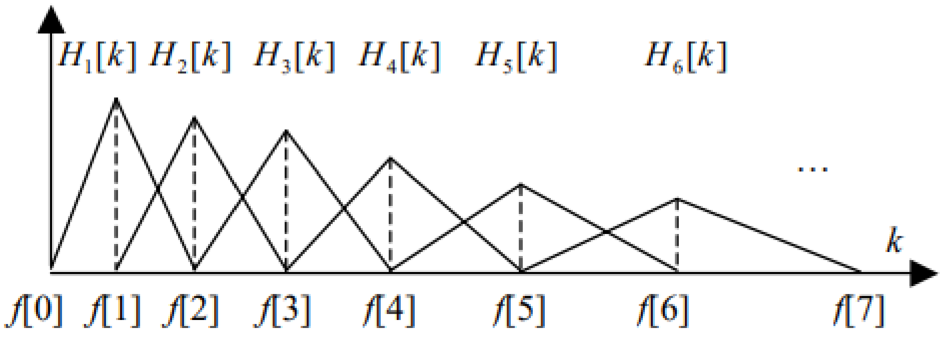


Figure 1.6. Triangular filters applied in mel-cepstrum calculation [9]

To conclude, feature extraction process is multistage operation, first step of which is sound analysis over short observation window. Every short analysis window is converted into spectral representation using FFT. Then, these spectrums are forwarded into Mel-Filters to acquire Mel scale Spectrum which undergoes cepstral analysis to obtain Mel-Frequency Cepstral Coefficients (MFCC). These coefficients represent input signal as a sequence of Cepstral feature vectors. Extracted feature vectors are extremely useful for the sound classification purposes.

1.5 Summary

Endowing machines with sound based situational awareness is a challenging problem which consists of numerous stages and requires structural approach. This chapter introduced the situational awareness with the emphasis on the sound-based perception and discussed the importance of application fields of sound aware systems. It also presented efficient digital signal processing techniques for extracting informative features from the sounds which can be effectively used as input to pattern recognition systems. This is essential preprocessing step which provides a basis for effective application of the state-of-art mechanisms to classify and categorize vast amount of input data. Next chapter takes a deep insight into the field of neural networks and their application to classification problems.

Chapter 2. DEEP LEARNING

Despite long history behind convolutional neural networks which were first introduced in the1980s, only after revolutionary work by Krizhevsky et al. [13] they have been widely accepted as a technique for various classification tasks. From that time, by substituting methods depending on manually engineered features convolutional neural networks created a room for enormous progress in multiple pattern recognition objectives, including traffic signs classification, handwriting and facial recognition, detection of pedestrians, natural language processing etc. Although the main application of this technology has been found in the contexts of visual recognition, convolutional neural networks have been also successfully applied in the analysis of music and speech [9] [14]. These attempts have demonstrated that methods seizing locality of data can produce viable solutions to issues faced in other areas. This chapter presents a thorough exposition of state-of-art deep learning techniques starting from trivial perceptron examples and culminating in complex deep network architectures.

2.1 Introduction to neural networks

Neural network is extremely powerful computational tool which can be used to approximate any highly sophisticated function. To do so, these networks should undergo process of training and be adjusted to specific problem. This section introduces the concept of neural networks, gives brief information about activation functions, describes neural network architecture and training process (including notion of cost and loss) and eventually provides backpropagation algorithm equations.

2.1.1 Multilayer perceptron

In the middle of last century, inspired by human brain Warren McCulloch and Walter Pitts proposed an idea of using simplified neurons for accomplishment of complex non-linear tasks [15]. Frank Rosenblatt developed this idea into artificial neuron called perceptron which at first stage was working only with binary input and outputs [16]. Figure 2.1 illustrates the simplest example of this neuron.

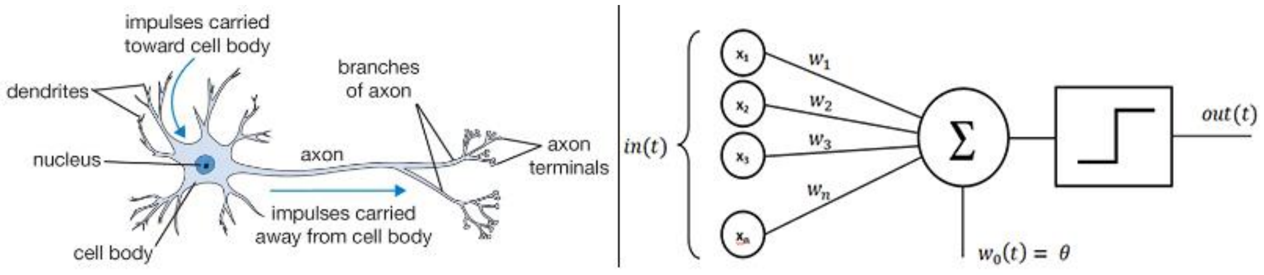


Figure 2.1. Simplest representation of biological neuron (left) and perceptron model (right) [17]

Similar to dendritic tree in human neurons taking as an input electrical signal and producing output in axons for further cells, perceptron gets input x_i, and produces output of logical 1 if the weighted sum of inputs is higher than threshold value and 0 otherwise.

$$output = \begin{cases} 0\ if\ \sum_i w_i x_i + b \le 0 \\ 1\ if\ \sum_i w_i x_i + b > 0 \end{cases} \tag{2-1}$$

$$z = \sum_i w_i x_i + b \tag{2-2}$$

Equations (2-1) and (2-1) summarize mathematical representation of perceptron. Here, w_i stands for weights vector, x_i stands for input vector and b is the negative threshold also known as bias. By adjusting weights and bias vectors during the process referred to as *network training*, one can attain required output for the given input.

Figure 2.2 demonstrates plain network also called *multilayer perceptron* (MLP) consisting of multiple neurons stacked together in three layers. From left to right, complexity of features extracted by each layer enlarges and network solves more abstract and sophisticated problems. However, there are obvious disadvantages of MLP

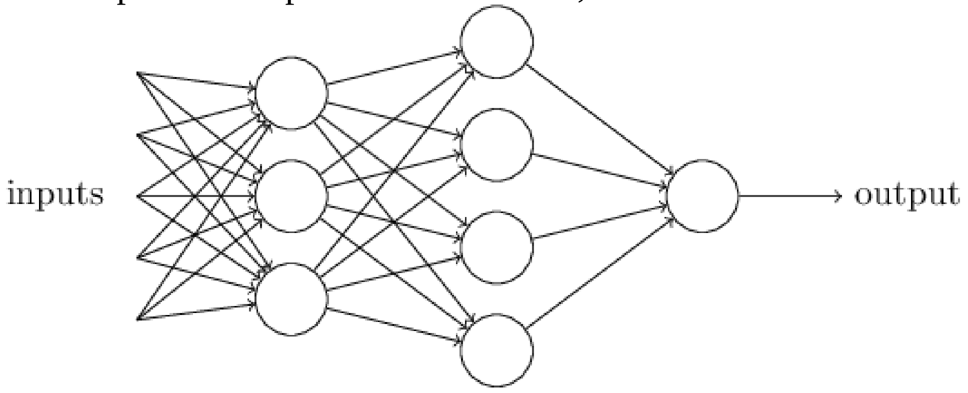


Figure 2.2. Multilayer perceptron [18]

in a presented form including linearity, inability to conduct multiclass classification and susceptibility to insignificant changes in bias or weight values, which can result in binary flip of inappropriate elements of the network. Consequently, MLP can be highly linear, sensitive to tiny variations and produce inaccurate output. To tackle these issues and make this system solve truly complex tasks, individual nodes should pass through *activation functions*.

2.1.2 Activation functions

Slightly modified version of perceptron includes non-linear activation function and is called artificial neuron. Activation functions define *firing rate* of the neuron and add required non-linearity to the network providing a basis for the development of sophisticated systems. There are multiple widely used activation functions including sigmoid (σ), hyperbolic tangent ($tanh$), rectified linear unit ($ReLU$) etc.

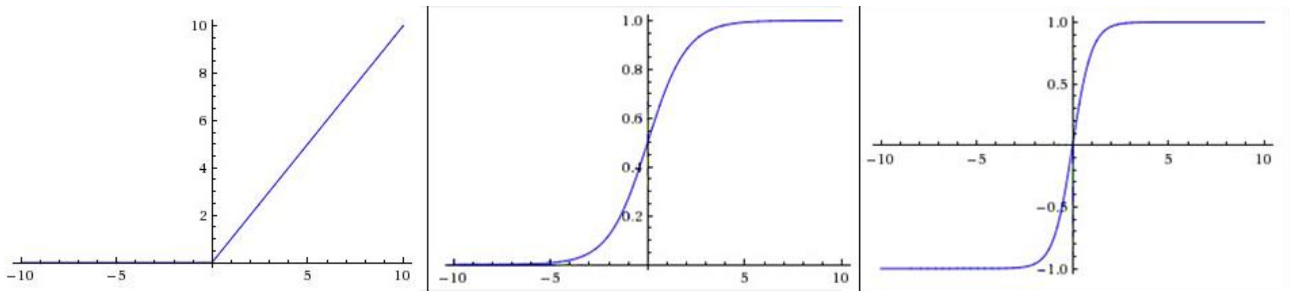


Figure 2.3. Activation functions. ReLU (left), sigmoid (middle), tanh (right) [17]

Sigmoid. A sigmoid or logistic activation function is a bounded real function which takes real-valued input and produces output in the range between 0 and 1 [17] [19]. Mathematical representation of sigmoid is given in equation 2-2.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2-3}$$

It is clear from (2-3) that this function takes values close to 1 whenever x is large positive number and 0 when it is large negative number.

Hyperbolic tangent. Unlike sigmoid function, $tanh$ squezees input into range between -1 and 1 which makes the output symmetrical around zero. Equation (2-4) demonstrates mathematical expression for $tanh$.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = 2\sigma(2x) - 1 \tag{2-4}$$

Since hyperbolic tangent is zero-centered, it is always preferred to sigmoid non-linearity because of its property to produce outputs (which are inputs for the next layer) that are on average close to zero [20]. This property is useful for backpropagation algorithm which is discussed later.

ReLU. Rectified linear unit function is increasingly gaining popularity in last few years due to high convergence rate during gradient descent process. Output of this activation function cuts the negative side of input and leaves only positive range. Equation (2-5) shows mathematical representation of ReLU.

$$f(x) = \max(0, x) \tag{2-5}$$

According to Krizhevsky et al. [13], rectified linear unit speeds up deep neural networks learning process six times compared to $tanh/sigmoid$ functions. This is due to the simple implementation of ReLU in contrast with expensive exponential calculations in the latter.

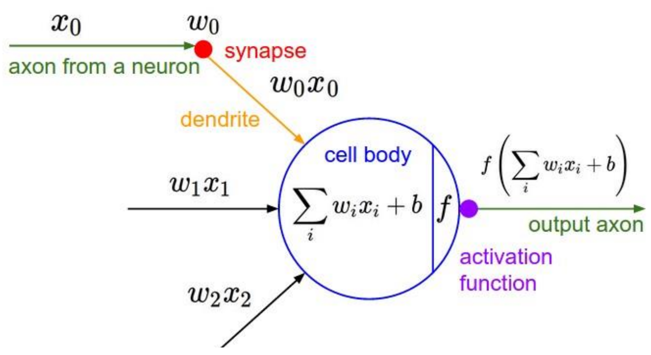


Figure 2.4. Artificial neuron model [17]

Figure 2.4 illustrates full model of artificial neuron. By drawing analogy with human brain cells, artificial neuron gets information from other neurons in dendrites (inputs x_i), computes their weighted sum, adds bias and passes through activation function which should be chosen by network designer. Equation (2-6) summarizes output vector prediction $\hat{y}$ of the neuron given input vector x.

$$\hat{y}^{(i)} = \sigma\left(w^T x^{(i)} + b\right) \qquad (2\text{-}6)$$

where $\hat{y}^{(i)}$ is the prediction of neuron for i-th example, $x^{(i)}$ is the i-th training example, w is the weights vector, b is bias vector and $\sigma\left(z^{(i)}\right) = \frac{1}{1+e^{-z^{(i)}}}$. The output of neuron is forwarded as input to the next layer or depending on architecture, reused (for example in recurrent neural networks) or taken as full output.

2.1.3 Neural network architecture

Although single neuron cannot produce useful computation and fit compound convex function, using output from stack of neurons as input for the next layer of neurons in sequence forms an artificial neural network which can produce highly non-linear functions. Figure 2.5 demonstrates typical simplified neural network architectures. The leftmost layer is referred as *input layer* which is opposite to the rightmost counterpart called *output layer*. All other layers in between are defined as *hidden layers*. It is noteworthy to mention that output of every node in any (except output) layer is directed as input to each neuron in successive layer. This architectural pattern is referred as *fully connected* or *dense* layer. While design and implementation of hidden layers is somewhat of art, decision on dimensions of input and output layers is straightforward. Number of neurons in input layer is equal to total size of input data. For instance, network for classification of grayscale images of size 64 by 64 pixels should have 4096 neurons in the input layer.

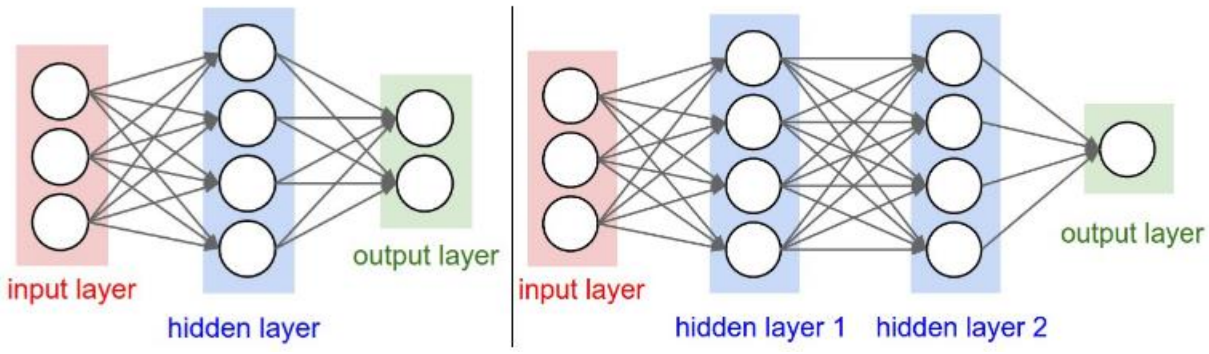


Figure 2.5. Neural network architectures: 2 layer (left) and 3 layer (right) [17]

For classification problems size of output layers is constrained by number of identifiable classes. For example, binary classification - simple decision between True and False requires only single neuron in the output. In contrary, distinguishing multiple categories of input (object classification, digit/letter recognition etc.) and giving probabilities for each of these classes requires output layer to be same size with number of possible categories. After taking decision on architecture one should start the process of *training* which in the simplest form can be explained on the example of logistic regression.

2.1.4 Training. Cost function. Logistic regression.

Appealing part of neural networks is that they can learn from input examples or in other words undergo the process of *training*. Recalling equation (2-6) which illustrates the relationship between input data and output of single neuron, it's obvious that having tuned proper weights and bias one can make a neuron act like linear classifier. Typical example of linear classification problem is drawing a *decision boundary* between distinct data classes. Figure 2.6 demonstrates trivial example of planar data classification. In order to draw separation line, it is required to determine weights and bias which accurately and optimally distinguish the input.

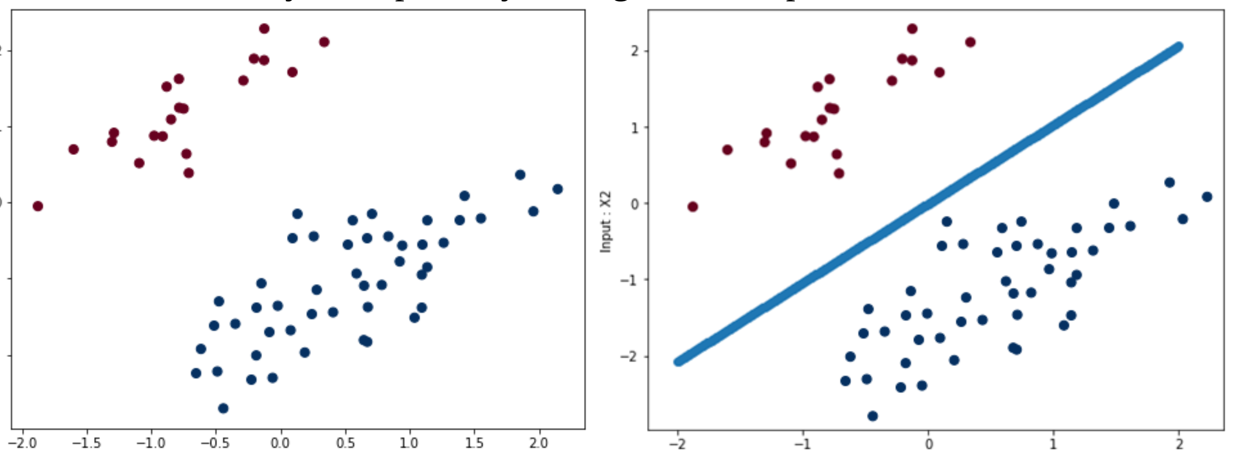


Figure 2.6. Planar data classification

The process of tuning these *parameters* is referred to as learning which represents a search through the space of possible hypotheses for one that will perform well, even on new examples beyond the training set [21, p. 695]. To qualify and evaluate how well this task is achieved one should use *loss function* (also referred to as objective function). There are various loss functions which determine how close neuron prediction is to the ground-true classification. In other words, they measure the difference between desired output y (from training examples) and prediction of neuron $\hat{y}$. Equation (2-7) demonstrates widely used cross-entropy loss function.

$$L(\hat{y}^{(i)}, y^{(i)}) = -\left(y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(\mathrm{i})}\right) \log\left(1 - \hat{y}^{(i)}\right)\right) \qquad (2\text{-}7)$$

Here $\hat{y}^{(i)}$ stands for prediction of the neuron for i-th example and $y^{(i)}$ refers to actual output value corresponding to the input. The choice of this function is justified by the fact that difference between neuron output and actual correct value should be minimized. Expression (2-8) shows mathematical expression for desired outcome of the neuron.

$$\begin{cases} \hat{y} = p(y = 1|x) \\ If\ y = 1{:}\ p(y|x) = \hat{y} \\ If\ y = 0{:}\ p(y|x) = 1 - \hat{y} \end{cases} \tag{2-8}$$

Here $p(y|x)$ defines the probability that output of neuron will be equal to the actual value in two different states. Apparently, $\hat{y}$ is the chance that output of neuron is equal to 1 when $y = 1$, and conversely, $1 - \hat{y}$ is the probability that output value is 0 when $y = 0$. Equation (2-7) perfectly matches these conditions if the activation function of neuron produces bounded output between zero and one. Average loss over the set of multiple training examples forms a *cost function*. Expression (2-9) defines cross entropy cost function which represents average loss of predictions over m training examples.

$$\begin{gathered} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) = \\ = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(\mathrm{i})}\right)\log\left(1 - \hat{y}^{(i)}\right)\right) \end{gathered} \tag{2-9}$$

The process of fitting weights of the model described by (2-6) to minimize cost function (2-9) on dataset containing m training examples is called *logistic regression* [21, p. 726]. Searching for optimal values of weights for minimizing the cost is a challenging problem which can be addressed by *gradient descent* algorithm.

2.1.5 Gradient descent

Having function of many variables which should be minimized one have to figure out efficient way to accomplish this task. *Gradient descent* is the most widely used optimization algorithm for searching a minimum of a function. Main idea of this method is taking proportional steps in the direction of negative steepest gradient of the function at given point.

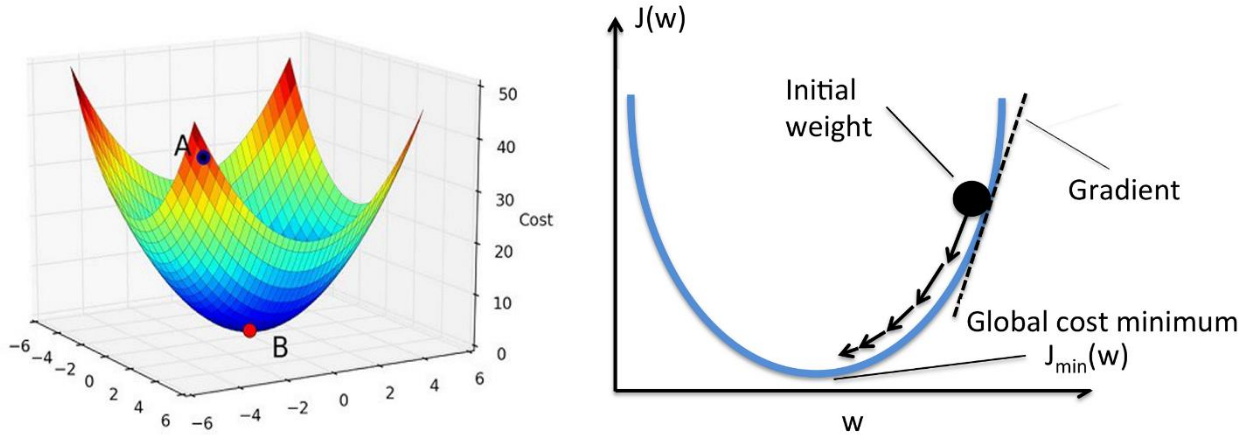


Figure 2.7. Multidimensional function to minimize (left), gradient descent of cost function in two dimensions (right)

Left side of Figure 2.7 demonstrates 3-dimensional representation of function to minimize. Starting from *initialization point* A one has to compute gradient and take a proportional step towards its negative direction until ultimately reaching minimum B. Right side of Figure 2.7 illustrates gradient descent applied to weight parameter of cost function in two-dimensional plane. Presented examples describe simplified representation of gradient descent, which in reality can be applied to multidimensional functions of many variables. Expressions (2-10) and (2-11) mathematically summarize gradient descent method applied to parameters of cost function $J(w,b)$ for a single training example.

$$w_{i+1} = w_i - \alpha \frac{\partial J(w,b)}{\partial w} \tag{2-10}$$

$$b_{i+1} = b_i - \alpha \frac{\partial J(w,b)}{\partial b} \tag{2-11}$$

Here w_{i+1} and b_{i+1} stand for the next generated point, α is *hyperparameter* of the training process also known as *learning rate* and $\frac{\partial J(w,b)}{\partial w}, \frac{\partial J(w,b)}{\partial b}$ are partial derivatives of cost function with respect to weights and bias accordingly. By reiterating these updates of parameters, one can “roll down a hill” and eventually find a minimum of a cost function [18].

2.1.6 Forward and backward propagation

Considering multidimensional nature of cost function, the most challenging part of gradient descent algorithm is the computation of partial derivatives. In order to efficiently calculate value of gradient, this operation has to be broken into two separate parts: forward and backward propagation.

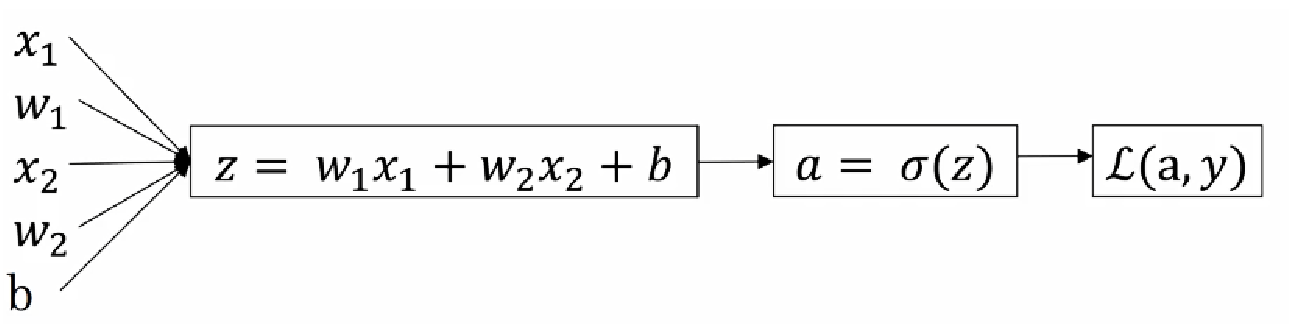


Figure 2.8. Computational graph of forward step

Figure 2.8 illustrates computational graph of the forward step on the example of logistic regression for single neuron with two inputs. Firstly, output of the neuron is obtained using expression (2-2), then one of activation functions such as sigmoid (2-3) is applied to this value and finally loss function is computed using equation (2-7).

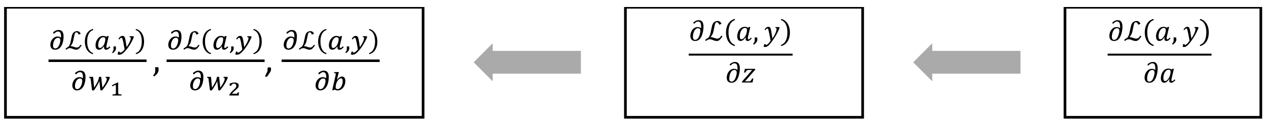

$\frac{\partial\mathcal{L}(a,y)}{\partial w_1}, \frac{\partial\mathcal{L}(a,y)}{\partial w_2}, \frac{\partial\mathcal{L}(a,y)}{\partial b}$
$\frac{\partial\mathcal{L}(a,y)}{\partial z}$
$\frac{\partial\mathcal{L}(a,y)}{\partial a}$

Figure 2.9. Computational graph of backward step

After obtaining value of the loss one can proceed to the calculation of gradients. Figure 2.9 illustrates computation graph for backward propagation step. Firstly, derivative of the loss function with respect to activation a is taken, which is expressed by equation (2-12) and derived by differentiating (2-7) .

$$\frac{\partial\mathcal{L}(a,y)}{\partial a} = da = -\frac{y}{a} + \frac{1-y}{1-a} \tag{2-12}$$

Following this, one can apply chain rule in order to obtain following expression:

$$\frac{\partial\mathcal{L}(a,y)}{\partial z} = \frac{\partial\mathcal{L}(a,y)}{\partial a} \odot \frac{\partial a}{\partial z} = da \odot g'(z) \tag{2-13}$$

$$\sigma'(z) = \sigma(z)(1-\sigma(z)) \tag{2-14}$$

Here da stands for partial derivative of loss function with respect to activation function and $g'(z)$ is derivative of activation function. It is noteworthy to mention that elementwise product is denoted by symbol $\odot$, which stands for Hadamard product. Assuming that the activation function is sigmoid derivative of which is given by expression (2-14), equation (2-13) can be rewritten as below:

$$\frac{\partial\mathcal{L}(a,y)}{\partial z} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right) \cdot a(1-a) = a - y \tag{2-15}$$

Expression (2-15) is crucial for describing partial derivatives with respect to neural network parameters, therefore it is usually denoted as "dz". Finally, derivatives with respect to bias and weights are obtained using following equations:

$$\begin{aligned}\frac{\partial\mathcal{L}(a,y)}{\partial w_1} &= \frac{\partial\mathcal{L}(a,y)}{\partial z}\cdot\frac{\partial z}{\partial w_1} = x_1 dz\\ \frac{\partial\mathcal{L}(a,y)}{\partial w_2} &= \frac{\partial\mathcal{L}(a,y)}{\partial z}\cdot\frac{\partial z}{\partial w_2} = x_2 dz\end{aligned} \qquad (2\text{-}16)$$

$$\frac{\partial\mathcal{L}(a,y)}{\partial b} = \frac{\partial\mathcal{L}(a,y)}{\partial z}\cdot\frac{\partial z}{\partial b} = dz \qquad (2\text{-}17)$$

By substituting expressions (2-16), (2-17) into gradient descent update rules (2-10), (2-11) and repeating this process multiple times, one can achieve minimum of the loss function and eventually train linear classifier to meet required specifications. However, application area of logistic regression is bounded to trivial problems, and it can not be utilized for solving complex issues. This is where neural networks come in to the scene.

2.1.7 Neural networks. Vectorization. Backpropagation

In the previous section basic forward and backward propagation steps were explained on the example of logistic regression. Figure 2.10 illustrates two examples of decision boundary problems, where logistic regression fails to fit complexity of data appropriately and demonstrates poor classification. However, as described in 2.1.3 multiple neurons stacked in various architectures or neural networks can handle much more sophisticated tasks. Training process of neural network is slightly more complex and involves vectorization which is essentially needed for replacing expensive computational loops with efficient matrix operations. Figure 2.11 demonstrates common notation used in deep neural networks.

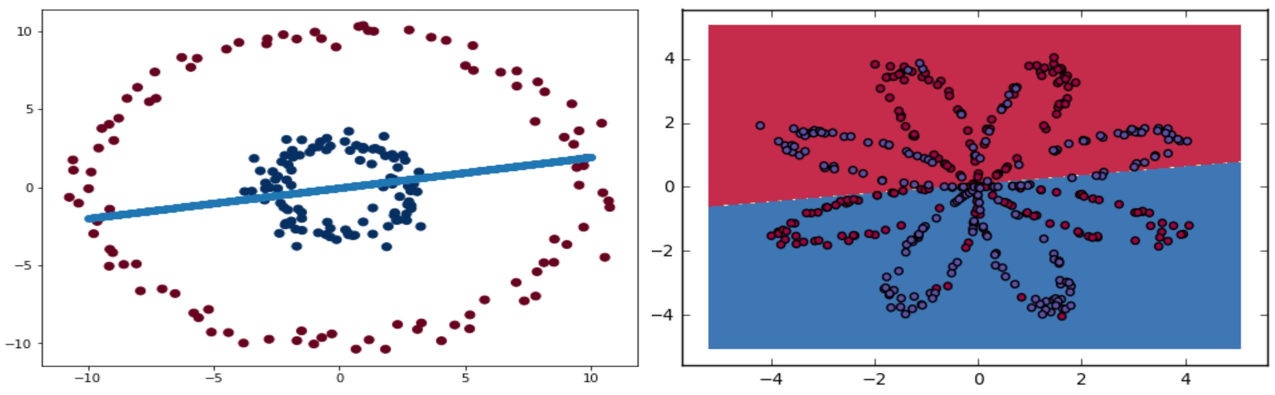


Figure 2.10. Examples of too complex problems for logistic regression

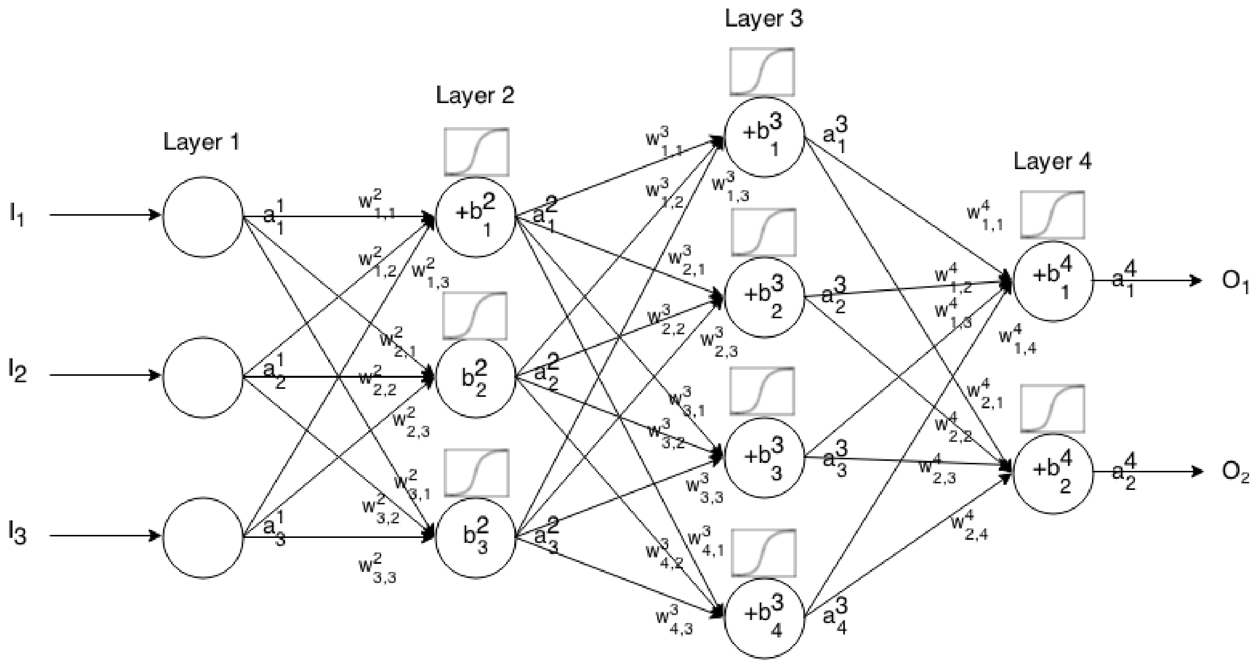


Figure 2.11. Neural network notations

Analogically to logistic regression (2-2) the output of the neuron in network is given by following expression:

$$a_j^{[l]} = \sigma\left(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}\right) \tag{2-18}$$

Here $a_j^{[l]}$ is the output of j^{th} neuron in layer l, σ is activation function, sum is over all neurons k in the $(l-1)^{th}$ layer, $w_{jk}^{[l]}$ is the weight from k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer, $b_j^{[l]}$ is the bias of j^{th} neuron in layer l [18]. Equation (2-19) presents vectorized form of activation output a of the neural network.

$$\begin{bmatrix} z_1^{[l]} \\ z_2^{[l]} \\ \dots \\ z_j^{[l]} \end{bmatrix} = \begin{bmatrix} \_\_\_ w_1^{[l]T} \_\_\_ \\ \_\_\_ w_2^{[l]T} \_\_\_ \\ \dots \\ \_\_\_ w_j^{[l]T} \_\_\_ \end{bmatrix} \begin{bmatrix} a_1^{[l-1]} \\ a_2^{[l-1]} \\ \dots \\ a_k^{[l-1]} \end{bmatrix} + \begin{bmatrix} b_1^{[l]} \\ b_2^{[l]} \\ \dots \\ b_j^{[l]} \end{bmatrix} \tag{2-19}$$

$$a^{[l]} = \sigma(z^{[l]}) \tag{2-20}$$

Here $a^{[l]}$ denotes vector of neurons output values in layer l, σ is activation function and $z^{[l]}$ stands for total weighted sum of inputs in layer l, including the bias term.

Figure 2.12 summarizes computational architecture of *backpropagation* algorithm which is used for adjusting synaptic parameters of the network. Obviously, one has to go through forward step in order to obtain value for the loss function and consequently by going back derive from previous step partial derivatives with respect to required parameters.

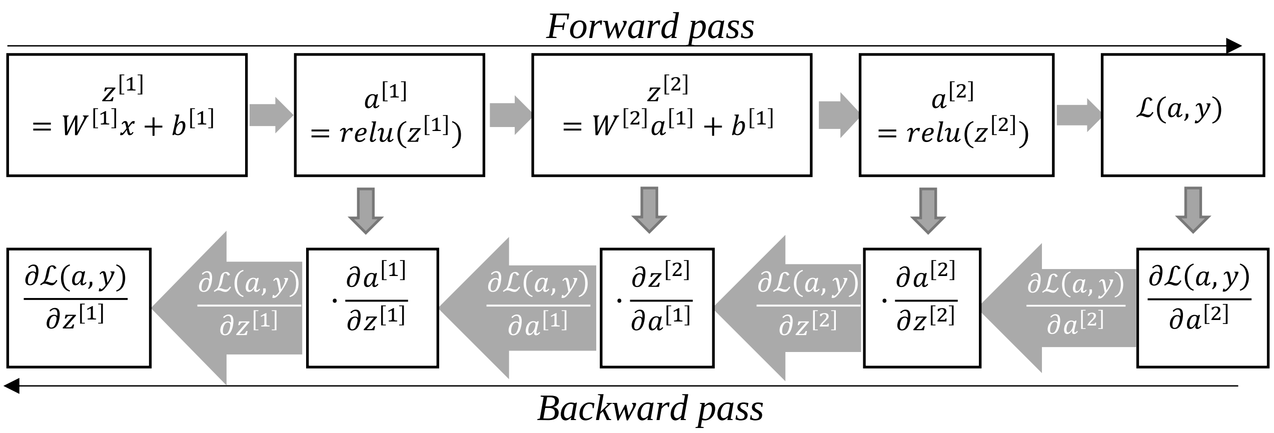

Forward pass
$z^{[1]} = W^{[1]}x + b^{[1]}$
$a^{[1]} = relu(z^{[1]})$
$z^{[2]} = W^{[2]}a^{[1]} + b^{[1]}$
$a^{[2]} = relu(z^{[2]})$
$\mathcal{L}(a,y)$
$\frac{\partial\mathcal{L}(a,y)}{\partial z^{[1]}}$
$\frac{\partial\mathcal{L}(a,y)}{\partial z^{[1]}}$
$\cdot\frac{\partial a^{[1]}}{\partial z^{[1]}}$
$\frac{\partial\mathcal{L}(a,y)}{\partial a^{[1]}}$
$\cdot\frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial\mathcal{L}(a,y)}{\partial z^{[2]}}$
$\cdot\frac{\partial a^{[2]}}{\partial z^{[2]}}$
$\frac{\partial\mathcal{L}(a,y)}{\partial a^{[2]}}$
$\frac{\partial\mathcal{L}(a,y)}{\partial a^{[2]}}$
Backward pass

Figure 2.12. Backpropagation on neural network

Table 1 provides fundamental equations of backpropagation by using which all partial derivatives within a neural network can be calculated. Left side of table show equations in a vector form, where gradients are calculated for a single training example. The derivation of these expressions come from logistic regression and the outcomes of chain rule differentiation.

Table 1. Backpropagation equations

Vector form	Matrix form
$dz^{[l]} = da^{[l]} \odot g^{[l]\prime}\left(z^{[l]}\right)$ (2-21)	$dZ^{[l]} = dA^{[l]} \odot g^{[l]\prime}\left(Z^{[l]}\right)$ (2-22)
$dw^{[l]} = a^{[l-1]T}dz^{[l]}$ (2-23)	$dW^{[l]} = \frac{\partial J}{\partial W^{[l]}} = \frac{1}{m}A^{[l-1]T}dZ^{[l]}$ (2-24)
$db^{[l]} = dz^{[l]}$ (2-25)	$db^{[l]} = \frac{\partial J}{\partial b^{[l]}} = \frac{1}{m}\sum_{i=1}^{m} dZ^{[l](i)}$ (2-26)
$da^{[l-1]} = w^{[l]T}dz^{[l]}$ (2-27)	$dA^{[l-1]} = \frac{\partial J}{\partial A^{[l-1]}} = W^{[l]T}dZ^{[l]}$ (2-28)

Equations (2-21), (2-22) and (2-23) are generalized across multiple layers in neural network and resemble (2-13), (2-16) and (2-17) from logistic regression accordingly. By taking closer look at backward step demonstrated on Figure 2.12, one can derive expression (2-27) using a chain rule as following:

$$\frac{\partial\mathcal{L}}{\partial a^{[l-1]}} = \frac{\partial\mathcal{L}}{\partial z^{[l]}}\,\frac{\partial z^{[l]}}{\partial a^{[l-1]}} = w^{[l]}dz^{[l]} \qquad (2\text{-}29)$$

Right side of Table 1 covers set of backpropagation equations for multiple training examples at once. This is achieved by further vectorization of equation (2-19), where each output column z for training example i is stacked to form a matrix Z. The same process is applied to activation vectors a and w to form matrixes A and W respectively. While training over multiple examples m one has to make two assumptions about the cost function J [18]. The first is that this function can be taken as an average of losses over all training examples. The second is that it can be written as a function of the outputs of the network.

Equations (2-22), (2-24), (2-26), (2-28) along with update rules (2-10), (2-11) summarize neural network training process and allow to progressively step towards minimum point of cost function (2-9). There are some crucial factors which have enormous influence on performance of training such as learning rate, choice of activation function, mini batch size, hidden units and hidden layers. Next section gives brief description of various optimization techniques and importance of tuning the values which are referred to as *hyperparameters*.

2.2 Neural network optimization. Hyperparameter tuning. Regularization

2.2.1 Stochastic, batch and mini-batch gradient descent

Previous section introduced training process over multiple examples at once, which is required for better generalization and performance of the network. However, passing too many examples into forward step of backpropagation algorithm in order to make a little update in network parameters may extremely slow down the training process. The process of complete pass through entire dataset in order to calculate the average gradient and make new descent step is referred to as *batch gradient descent* [20]. In contrary, one can compute an estimate of true gradient based on single random example chosen from dataset and then update the weights and bias. This process is named *stochastic (online) gradient descent* (SGD).

Table 2. Comparison of stochastic and batch gradient descents

Stochastic gradient descent	**Batch gradient descent**
+Updates much faster than batch training	+Convergence conditions are well defined
+Results in better solutions	+Many acceleration techniques applicable
-Loses speed from vectorization	-Too slow update rate

Table 2 presents advantages and drawbacks of each aforementioned approaches. Although stochastic gradient descent is fast at making updates to the weights, training on single example deprives the possibility to take advantage of optimization techniques such as vectorization. In contrast efficiency of batch gradient descent can be enlarged in numerous ways, however network updates are done only after processing of entire dataset, which may lead to excessive speed losses.

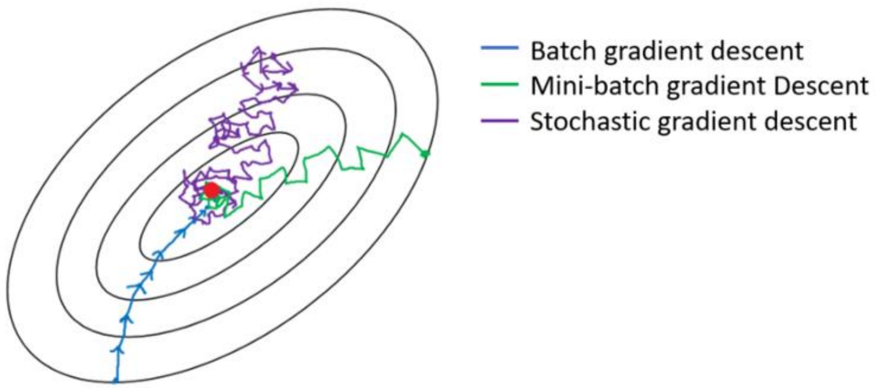


Figure 2.13. Learning paths using stochastic, batch and mini-batch gradient descents [22]

Mini-batch gradient descent takes advantages of both approaches by making updates in the network based on mini-batch of training examples. It reduces variance of parameter updates which leads to more stable convergence compared to stochastic gradient descent and enables application of matrix optimizations [23]. Another advantage is that there is no need to pass through entire training dataset in order to make a progress, which extremely boosts training speed.

Figure 2.13 demonstrates comparison of typical learning paths of all approaches. It is obvious that batch GD may take confident large steps towards the minimum of the cost function. Oppositely, stochastic gradient descent is subjected to high noise and continuously fluctuates around minima point never reaching it. The mini-batch gradient descent moves towards extremum by taking steps which are not always headed towards minima point but more consistent than previous step. Number of training examples in a single batch define another hyperparameter known as *mini-batch size*. Usually it is taken to be power of 2 in order to increase computational efficiency of gradient descent. Typical sizes are 64, 128, 256 and 512 examples per mini-batch.

2.2.2 Train/Dev/Test sets

Quality and performance of neural network is extremely dependent on the input data used for training and validation. It is the data where complexity and magic of a network come from and feeding in appropriately labeled input determine how successful network is at performing a certain task. Before training and measuring the efficiency of the network, one has to divide the whole data in three separate sets. First is obviously a *training set* which consists of examples used for learning, that is to fit the parameters of the classifier [24]. After training, the network is evaluated on a *test set* which represents a set of examples used only to assess the performance of a fully specified classifier [24]. It is vital that network does not encounter any example from test set during the training. Finally, dev (development or validation) set is used while training for comparison of various hyperparameter values and several network parameters. To summarize, during the training of a network on training set one has to validate different models on a dev set and after decision on most efficient architecture evaluate it on the test set. Crucially, all three datasets should follow the same probability distribution and contain examples from all classes (in case of classification problem). Figure 2.14

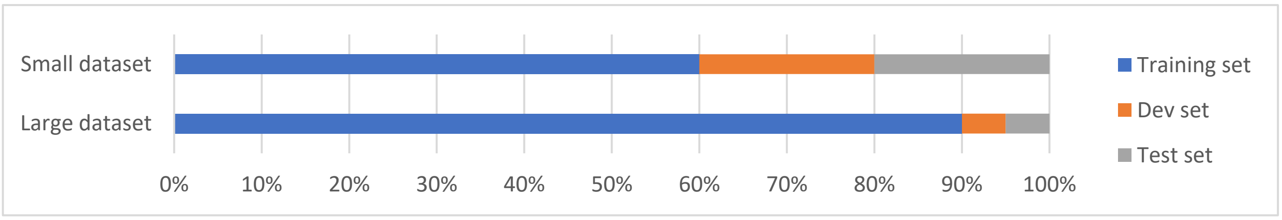


Figure 2.14. Distribution of training/dev/test sets

illustrates typical ratio for splitting data into these three sets. When the dataset is large (say 10^6 examples) its main portion should be utilized for training in order to generalize network well while a small partition of 0-5% can be used for test and validation. However, dev and test fractions of small datasets should be larger to assure that validation is done properly.

2.2.3 Bias and variance

In any practical application of neural networks, the aim of training is not to resemble and learn exact form of training data, but instead to build a statistically generalized model which exhibits well on unprecedented inputs. Training a neural network model which has too inflexible and simple architecture may *underfit* the data and introduce a high *bias*. In contrast too complex and sophisticated model *overfits* the data and demonstrates high *variance*. Figure 2.15 illustrates graphical representation of models having these deficiencies. Obviously, the model in the left side demonstrates poor classification and is unable due to simplicity correctly characterize the data. Oppositely, rightmost model fits data too much and shows insufficient generalization. The model in the middle accomplishes the task just right and indicates good compromise between bias and variance.

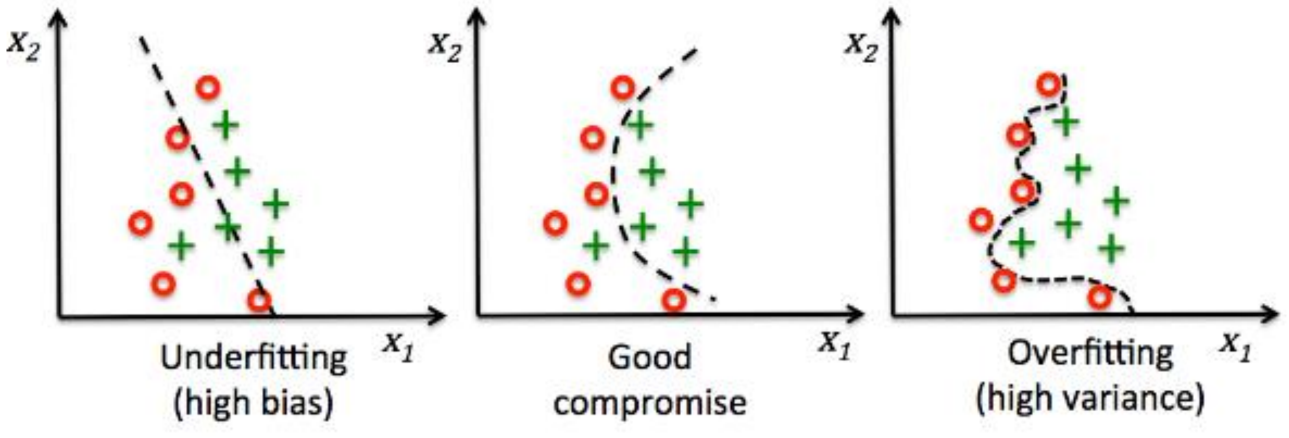


Figure 2.15. Models with high bias(left), high variance (right) and good compromise (middle) [25]

After assessment of neural network accuracy on training and dev set it is possible to make conclusions about efficiency of classification and determine whether the model is successful or has high variance, high bias or both. It is clear from first column in

Table 3 that model performing well on training set and failing to do so on dev set is subjected to overfitting and has high variance. If the value of error is considerable, but the gap between training and dev sets performance is small, (like in the third column of Table 3) it points out to high bias and underfitting of model. The worst-case scenario is when model has large training error and even greater dev set error which indicates that there are both high bias and variance. In contrast, last column of Table 3 demonstrates optimal case when both training and dev set errors are insignificant.

Table 3. Training/dev sets evaluation errors for various models

Training set error	2%	14%	16%	0.6%
Dev set error	13%	15%	33%	1%
Conclusion	High variance	High bias	High bias and variance	Good compromise

The generalization is maximized when there is a best compromise between the conflicting requirements of small bias and small variance [26, p. 332]. There are numerous methods referred to as *regularization techniques* which allow to reduce overfitting, smooth out the classifier and increase generalization of the network.

2.2.4 L2 Regularization. Weight decay.

Previous section introduced neural network training issues related to overfitting and underfitting. For reaching optimal equilibrium between bias and variance one has to figure out a methodology to regulate the effective complexity of the classifier. One of the principal approaches to accomplish this is application of *regularization techniques* which involve addition of penalty term to the error function [26]. Equation (2-30) mathematically describes one of such techniques which is referred to as L2 regularization also known as *weight decay*.

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m}\sum_{w} w^2 \qquad (2\text{-}30)$$

Here J stands for cost function of neural network consisting of sum of total losses $\mathcal{L}$ over m training examples and penalty term $\frac{\lambda}{2m}\sum_{w} w^2$ which represents a sum of the squares of all the weights in matrix w multiplied by λ – referred to as *regularization parameter* [18]. With this modification update rule for weights (2-10) for m training examples transforms into equation (2-31).

$$w_{i+1} = w_i - \frac{\alpha}{m}\sum_i \frac{\partial J(w,b)}{\partial w} - \frac{\alpha\lambda}{m} w_i = \\ = \left(1 - \frac{\alpha\lambda}{m}\right) w_i - \frac{\alpha}{m}\sum_i \frac{\partial J(w,b)}{\partial w} \tag{2-31}$$

This expression resembles usual update rule for weights with the only exception of $\left(1 - \frac{\alpha\lambda}{m}\right)$ weight decay factor. The generalization effect of L2 regularization is explained by the fact that it enforces network to keep weights small which makes it more robust to small changes in input and prevents from learning local noise in the data. Looking at the depth of the issue, regularization constrains neural network to fit into relatively simplified models based on frequent patterns appearing in the training set and persistent to overfitting resulted by learning specific characteristics of the noise.

2.2.5 Dropout regularization

Another powerful and crucial technique for reducing overfitting is called dropout and refers to disabling certain distribution of hidden and input units during the training. Figure.2.16 schematically illustrates this procedure. Each unit is retained with a fixed probability p independent of other units, where p can be chosen using a validation set or can simply be set at 0.5, which seems to be close to optimal for a wide range of networks and tasks [27].

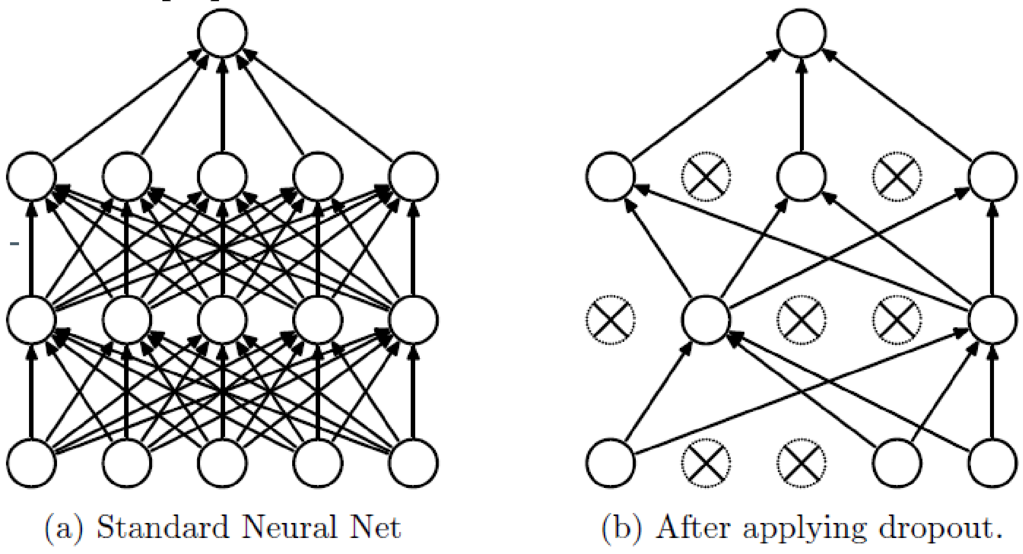


Figure.2.16. Standard neural net (left). Thinned net produced by application of dropout (right) [27]

After application of the dropout to the neural network, feed forward operation described by (2-2) transforms into expression (2-32).

$$\begin{aligned} r_j^{[l]} &\sim Bernoulli\ (p), \\ \tilde{y}^{[l]} &= r^{[l]} * y^{[l]} \\ z_i^{[l]} &= w_i^{[l]} \tilde{y}^{[l-1]} + b_i^{[l]} \end{aligned} \tag{2-32}$$

Here $r^{[l]}$ stands for vector of independent Bernoulli random variables each of which has a probability p of being 1 and * denotes elementwise multiplication. Thinned outputs $\widetilde{y}^{[l]}$ are produced by multiplication of output of layer $y^{[l]}$ by the vector p which is referred to as *dropout rate* and this process is consequently repeated at each layer. Due to random drop out of hidden units, neural network can not rely on a single feature and spreads out the weights over the whole net. This regularization technique enlarges efficiency of neural network and considerably reduces overfitting in a wide range of applications including pattern recognitions and classification.

2.2.6 Training optimization. Gradient descent with momentum. RMSprop. Adam optimization algorithm

There are numerous extensions to gradient descent algorithm which considerably optimize training speed and lead to faster convergence. Due to susceptibility to noise gradient descent usually takes steps which are not in the optimal direction and thus learning process can be highly oscillatory. Adding momentum to the parameter update procedure accelerates the steps towards relevant direction and smooths fluctuations.

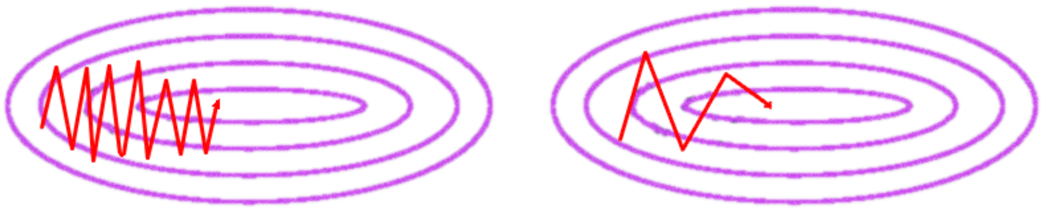


Figure 2.17. Gradient descent with momentum

By computing exponentially weighted average of previous gradient updates and applying fraction of it for the calculation of new gradients, one can easily add a momentum to reduce the oscillations. Expression (2-33) provides mathematical formulation of the gradient descent update rules with momentum.

$$\begin{aligned} v_{dW} &= \beta v_{dW} + (1-\beta)dW \\ v_{db} &= \beta v_{db} + (1-\beta)db \\ W_{i+1} &= W_i - \alpha v_{dW}, b_{i+1} = b_i - \alpha v_{db} \end{aligned} \tag{2-33}$$

Here v_{dW} and v_{db} stand for exponentially weighted averages of gradients, β is *momentum* parameter which defines smoothing intensity and α is learning rate. Figure 2.17 illustrates speed up gained by application of momentum. RMSprop algorithm accomplishes resembling task to gradient descent with momentum by taking moving average of squared gradient for each weight *[28]*. Expression (2-34) similarly modifies gradient update rules by taking exponentially weighted average of quadratic gradient and then dividing it by square root of mean.

$$\begin{aligned} s_{dW} &= \beta s_{dW} + (1-\beta)dW^2 \\ s_{db} &= \beta s_{db} + (1-\beta)db^2 \\ W_{i+1} &= W_i - \frac{\alpha dW}{\sqrt{s_{dW}}}, b_{i+1} = b_i - \frac{\alpha db}{\sqrt{s_{db}}} \end{aligned} \tag{2-34}$$

Finally, one of most widely used algorithms which has demonstrated high performance in numerous domains combines gradient descent with momentum with RMSprop and is called Adam optimization algorithm which stands for adaptive moment estimation. Typical implementation of Adam optimizer is defined as follows:

$$v_{dW} = \beta_1 v_{dW} + (1-\beta_1)dW \qquad s_{dW} = \beta_2 s_{dW} + (1-\beta_2)dW^2$$
$$v_{db} = \beta_1 v_{db} + (1-\beta_1)db \qquad s_{db} = \beta_2 s_{db} + (1-\beta_2)db^2$$
$$v_{dW}^{Corr} = \frac{v_{dW}}{(1-\beta_1^t)} \qquad s_{dW}^{Corr} = \frac{s_{dW}}{(1-\beta_2^t)}$$
$$v_{db}^{Corr} = \frac{v_{db}}{(1-\beta_1^t)} \qquad s_{db}^{Corr} = \frac{s_{db}}{(1-\beta_2^t)}$$

$$W_{i+1} = W_i - \alpha \frac{v_{dW}^{Corr}}{\sqrt{s_{dW}^{Corr}} + \varepsilon}, \qquad b_{i+1} = b_i - \alpha \frac{v_{db}^{Corr}}{\sqrt{s_{db}^{Corr}} + \varepsilon} \tag{2-35}$$

Here, firstly exponentially weighted averages and RMSprop are undergoing bias correction procedure and then they are combined in equation (2-35) which summarizes Adam optimization algorithm.

To summarize, this section introduced various optimization and error correction techniques for training neural networks. The next section focuses on convolutional neural networks and deep learning architectures.

2.3 Convolutional neural networks. Deep Learning

Up until this point adjacent layers of neural networks presented here were fully connected to each other (Figure 2.5). Although dense networks decently cope with such tasks as character recognition, there are several considerable issues with this architecture. Firstly, taking an example of image recognition, where there is a massive amount of inputs to the network, one can conclude that colossal number of weights needs to be trained which results in necessity of high computational and memory resources due to the growth in the network capacity.

Another primary disadvantage of the fully connected networks for pattern recognition applications is the absence of invariance with respect to translation and local distortion of the input *[29]*. Lastly, unstructured networks completely ignore the topology of the input, that is variations in the input arrangement do not affect the result of the training. In contrast, spectrogram representations of sounds have strong correlations between spatially adjacent variables. These correlations allow to take advantage of feature extraction and combination before classification of spatial objects, because neighbouring inputs can be clustered into small categories (corners, edges). Yann LeCun et al. developed idea of using *convolutional neural networks* which tackle aforementioned issues by combining three architectural ideas including shared weights, local receptive fields and spatial or temporal sub-sampling also referred to as pooling *[29]*.

2.3.1 Local receptive fields. Shared weights.

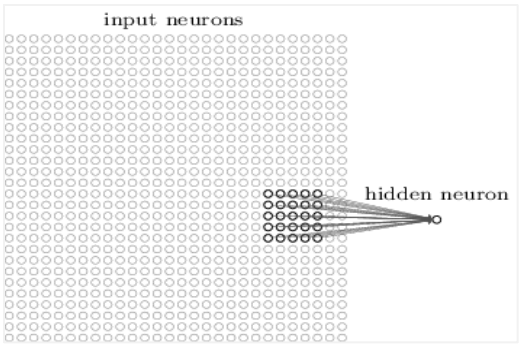


Figure 2.18 Local Receptive field [18]

Typical architecture of convolutional neural network implies connection of every neuron in the initial hidden layer to the little area in the input layer. This small region is referred to as *local receptive field* (Figure 2.18) for the hidden neuron [18]. This architecture enables the neurons to retrieve simple *features* of images including edges, corners and endpoints or corresponding features in the time-frequency representations of sounds. Combinations of these features allow to recognize higher-order concepts in successive layers. It turns out that simple feature detectors which are beneficial in one part of the input, can also be applied throughout the whole input signal. This is achieved by the fact that each neuron in the single plane of hidden layer *shares identical weights and bias* vectors with all other neurons. The output of this unit plane is referred to as a *feature map* and it formed by sliding and convolving the local receptive field with weight matrix over the whole area of the input. *Shared weights* and *bias* vectors for the single feature map leads to conclusion that all neurons in this map detect the same feature but in the different locations of the input signal. Together these parameters are said to define a *filter* also referred to as *kernel*. Number of filters applied to single layer define the number of output feature maps, while their size determines the dimensions of generated feature map.

$$n \times n \quad * \quad f \times f \quad \rightarrow \quad (n - f + 1) \times (n - f + 1) \qquad \text{(2-36)}$$

Expression (2-36) summarizes transformation of the network dimensions for the convolution operation between input layer of size n and filter of size f which is referred

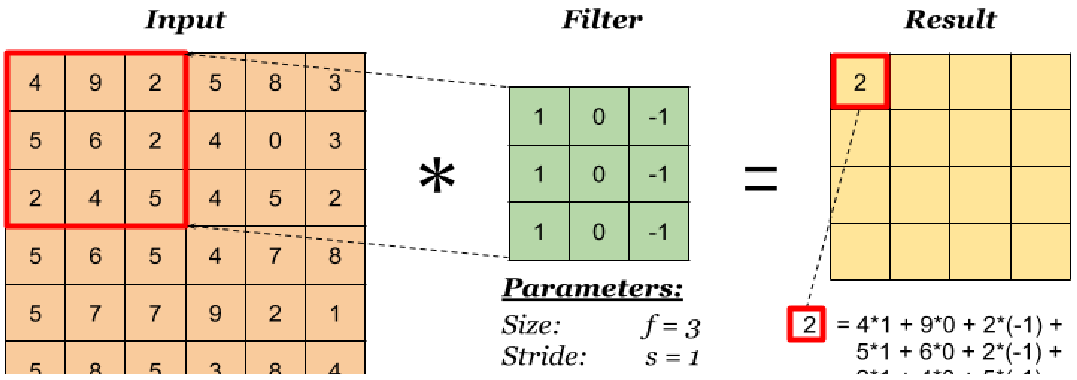


Figure.2.19. Single convolution operation

to as *valid* convolution. Figure.2.19 schematically illustrates mathematical procedure behind the single convolution operation on the example of vertical edge detector. Each element in local receptive field of input is multiplied with corresponding value in weight matrix and summed with other weighted inputs. After adding a bias, total value is passed through activation function (2-37).

$$relu\left(b + \sum_{l=0}^{f}\sum_{m=0}^{f} w_{l,m} a_{j+l,k+m}\right) \tag{2-37}$$

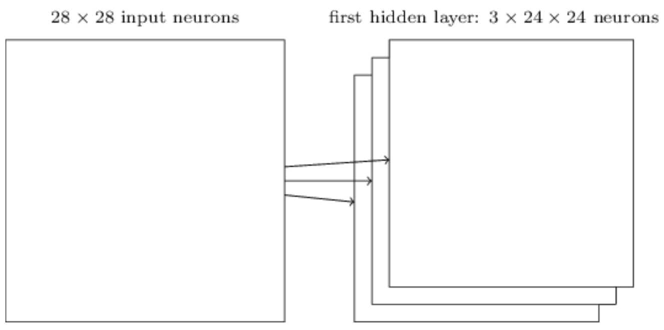


Figure 2.20 Feature map with 3 filters [18]

A complete layer in convolutional neural network may consist of multiple feature maps (each with different bias and weights vector) so that it is possible to extract multiple features at each position. Figure 2.20 illustrates example of 3 feature maps generated by the filter of size 5x5 from 28x28 input field. This layer can determine 3 various features with each feature identifiable throughout the whole input. Great advantage of the convolutional neural networks is the reduced number of parameters needed for efficient computation and classification. In the example above, for generation of each feature map only 5x5=25 shared weights and 1 bias parameter is needed which totals into 26x3=78 total parameters which is considerably lower than fully connected layer which even with 3 hidden nodes requires 28x28x3 weights and 3 biases totaling into 2355 parameters.

2.3.2 Stride and padding

Figure 2.21 demonstrates the process of feature map formation which is achieved by taking convolutional sum over certain region, putting this value into the feature map and then repeating the procedure after sliding the window further. The size of sliding step, defined as *stride*, determines the dimensions of the generated feature map. Larger stride results in smaller feature map, because the convolutions are taken by skipping certain intervals of the input.

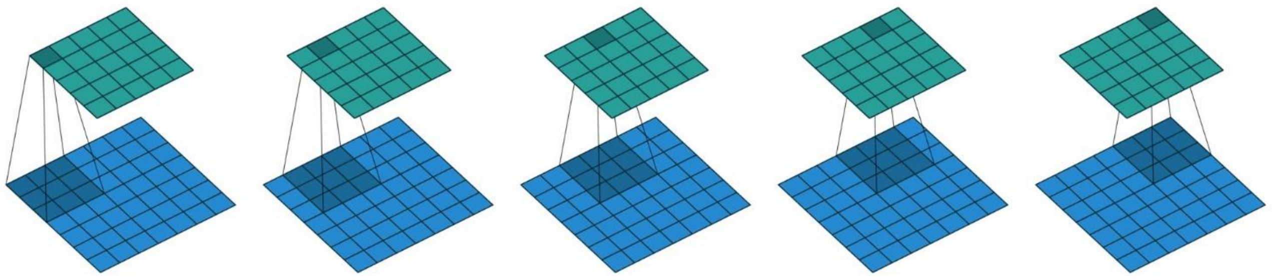


Figure 2.21. Valid convolution operation with stride=1, no padding and filter size=3x3 over the input layer with dimensions 7x7 [30]

In order to make the size of feature map same as the input, one has to apply zero padding that it is to pad input with zeros to keep the original dimensions after convolution. This operation is referred to as *same* convolution and depicted on Figure 2.22.

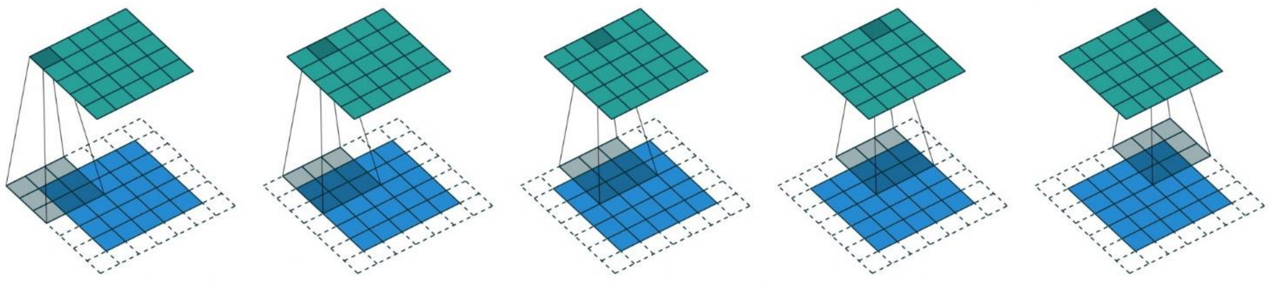


Figure 2.22. Same convolution operation with stride=1, filter size=3x3, padding=1 over the input layer with dimensions 5x5 [30]

Expression (2-38) modifies previously defined (2-36) to consider padding and strides for determination of new dimensions.

$$n \times n \quad * \quad f \times f \xrightarrow[\substack{stride\ s \\ padding\ p}]{} \frac{n+2p-f}{s}+1 \ \times \ \frac{n+2p-f}{s}+1 \qquad \text{(2-38)}$$

2.3.3 Convolutions over volume. Single convolutional layer

Section 2.3.1 introduced convolution operation over single layer input, however, almost always the inputs in CNN are given by several levels which are called *channels*. In this case filter should have the same number of channels as the input in order to properly match the convolution dimensions. Figure 2.23 demonstrates convolution operation over volume on the example of RGB image and vertical edge detector working in three different channels.

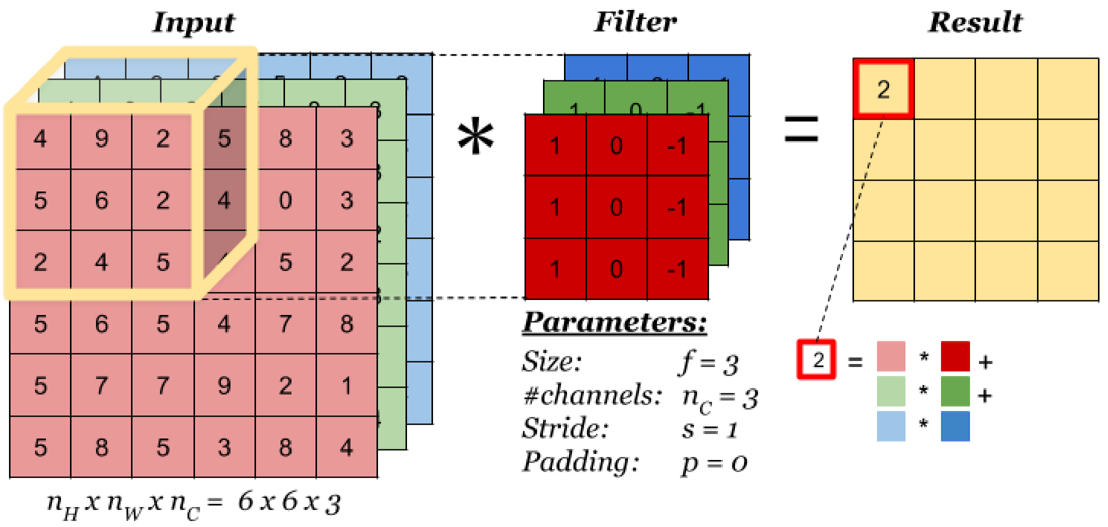


Figure 2.23. Convolutions over volume [31]

The idea of the convolution here is the same as in the single dimension case, with only difference in that convolutional window has 3 dimensions. Carrying out convolution with multiple filters results into the multidimensional feature map (Figure 2.24) similarly to the procedure illustrated on Figure 2.20.

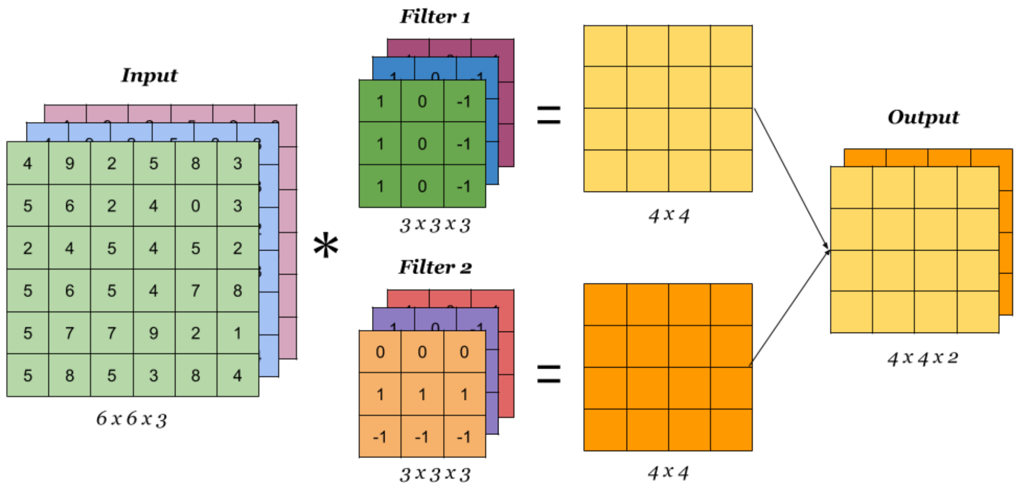


Figure 2.24. Convolutions over volume with multiple filters [31]

To summarize, it is crucial to define a notation which allows to appropriately specify dimensionality of each layer and correctly consider each parameter.

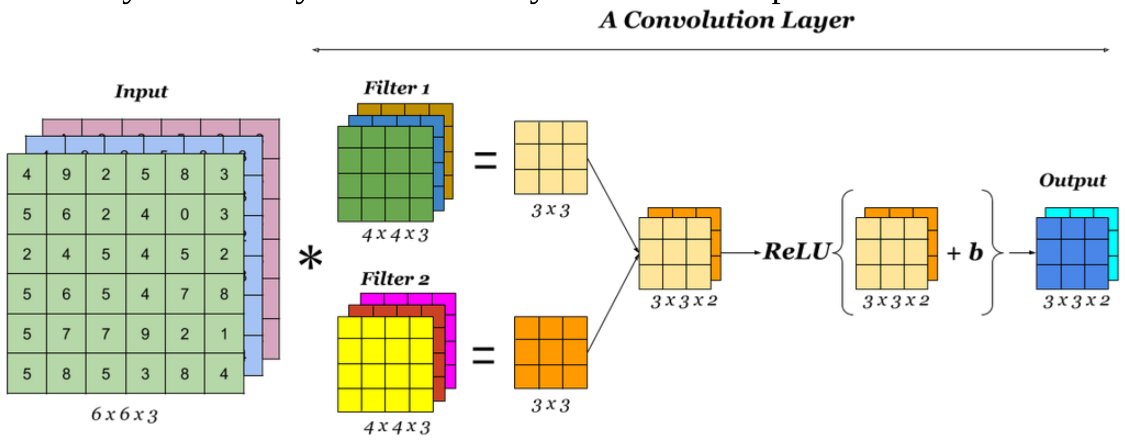


Figure 2.25 Single convolutional layer [31]

A complete **single convolutional layer** conducts cross correlation operation over the input and all set of filters defined by the architecture of the network. The outcomes of this procedure are stacked together to form feature maps, dimensions of which are updated according the rules presented in the Table 4. Eventually, these feature maps are passed through activation function to produce the final output. Figure 2.25 summarizes structure of single convolutional layer.

Table 4. Notations and dimension update rules for a single convolutional layer

$f^{[l]}$- filter size in layer l $p^{[l]}$- padding size in layer l $s^{[l]}$- filter size in layer l $n_c^{[l]}$- filter size in layer l Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$ Activations: $a^{[l]} \rightarrow n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$	If input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$ Then output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$ $n^{[l]} = \left\lfloor \frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$ $A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

2.3.4 Maximum and average pooling

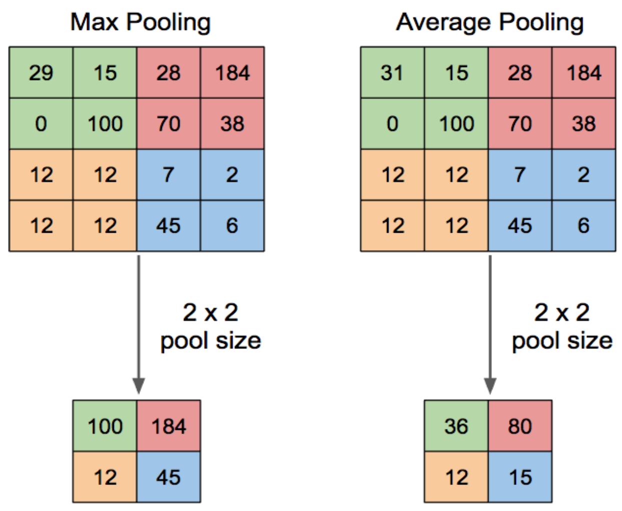


Figure 2.26. Maximum and average pooling [31]

In convolutional neural networks it is crucial to decrease spatial resolution of the feature map in order to diminish the precision with which the position of the specific features is encoded and thus make the network more robust to detection of this features regardless of their spatial position [29]. This can be accomplished by *sub-sampling* or *pooling* layers which carry out either local averaging or maximization, reducing dimensions of feature map and susceptibility to distortions and shift. Figure 2.26 demonstrates the examples of maximum and average pooling with pool size of 2x2 and stride of 2. In the case of maximum pooling the peak values from small local area in the input are taken to the next layer, while average pooling transmits the average of the pooling window. It is noteworthy to mention that pooling layers do not have learnable parameters and serve only for reducing the dimensions of the feature map following expression (2-38) without padding ($p = 0$).

2.3.5 Deep convolutional neural network architectures

At this point it is possible to take a look at some of the classical deep convolutional architectures. **LeNet-5** is one of the first examples of deep convolutional networks and it was proposed by the pioneer of this field Yann LeCun *et* al. in the work related to handwritten digit and letter recognition. He suggested focusing on training the feature extractor using convolutional layers which then pass extracted features to the fully connected classifier in the end. Figure 2.27 demonstrates the architecture of the LeNet-5 consisting of 2 convolutional layers followed by average pooling and passing features to 2 fully connected layers at the end of the network.

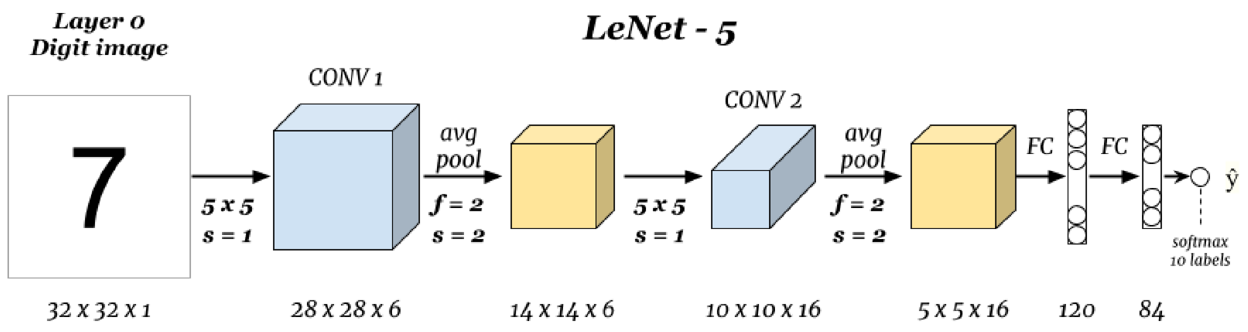


Figure 2.27. Architecture of LeNet-5 [31]

It is noteworthy to mention that dimensions of feature maps from left to right is reduced while the number of filters (or the depth of network) is gradually growing. Although Lenet-5 did well for recognition of grayscale images of size 32x32 showing only 0.95% of error on the test set, lack of computational power at the time of publishing the paper did not allow to make recognition of high dimensional images.

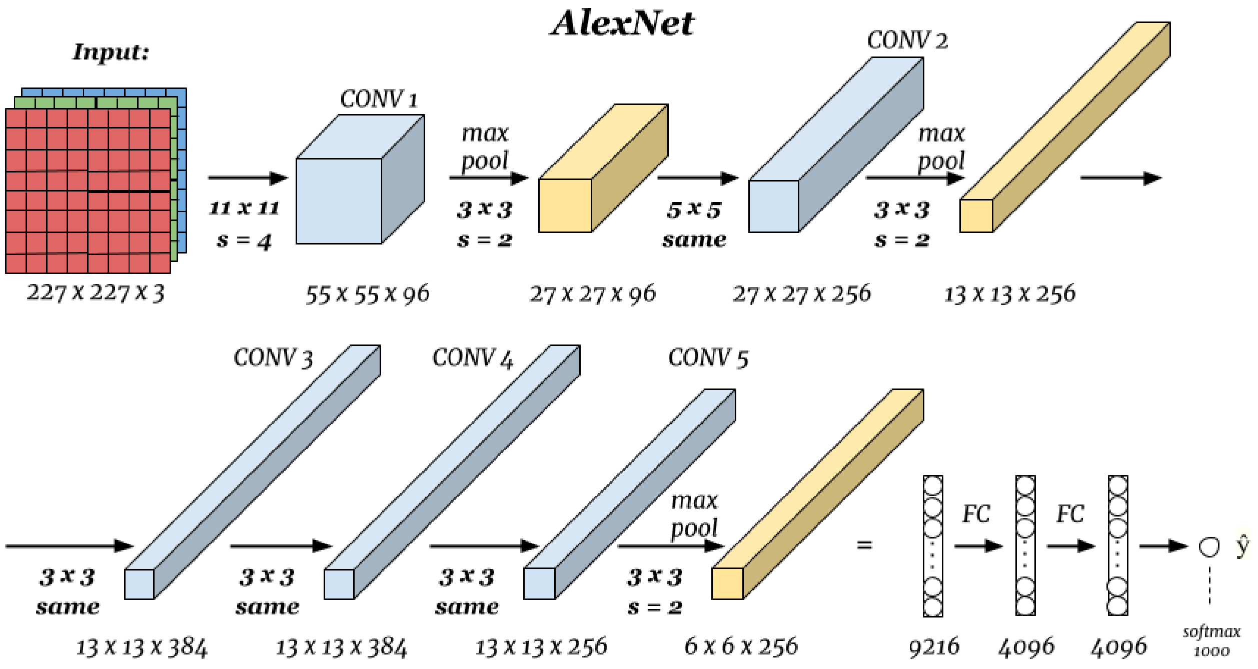


Figure 2.28. AlexNet architecture [31]

True revolution in the field of deep learning occurred after introduction of **AlexNet** architecture by Alex Krizhevsky *et al.* in 2012 [13]. It was designed to classify 1000 different classes of images from millions of examples in ImageNet dataset and required a large learning capacity. Figure 2.28 summarizes the architecture of AlexNet which consists of five convolutional and three fully connected layers. 1000-way softmax classifier is added to the output of the last dense layer in order to breakdown it into 1000 class labels. The input image of size 224x224x3 is filtered by 96 kernels of size 11x11x3 with a stride of 4 pixels in the first convolutional layer and passed through max pooling. The output of the pooling is fed into the second convolutional layer which produces 256 feature maps using kernels of size 5x5x48. There is no any pooling or normalizing layer between third, fourth and fifth convolutional layers. Third layer uses 384 filters of size 3x3x256 and passes output into fourth layer having 384 kernels each of size 3x3x192. Finally, fifth layer closes up the conv nets with 256 kernels of size 3x3x192 followed by fully connected layers containing 4096 neurons. AlexNet was extremely successful at the time of the publication and reached 37.5% test set error which was setting state-of-art result. However, despite high performance of this architecture it is too complicated and contains a lot of hyperparameters to tune (layer dimensions, kernel sizes, pooling, activations, dropouts etc.). **VGG-16** architecture simplified the concept of deep network by switching all filters in convolutional layers to the size of 3x3 with stride of 1 and taking all pooling layers as max with kernel size of 2x2 and stride of 2 [32]. Complete architecture of VGG-16 which consists of 13 convolutional and 3 fully connected layers is presented in Figure 2.29.

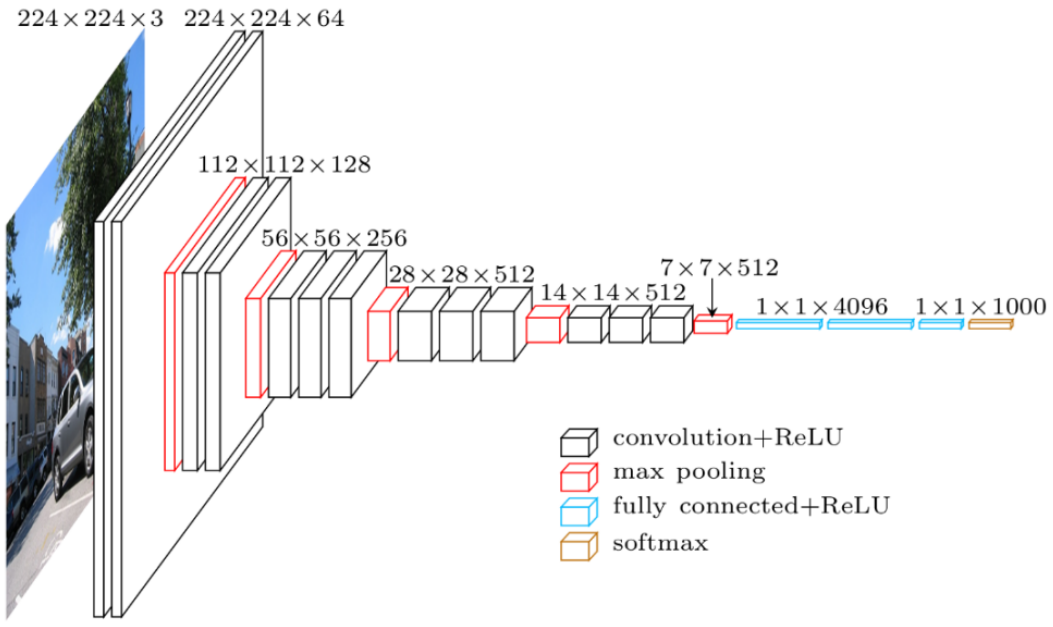


Figure 2.29. VGG-16 architecture [31]

It is noteworthy to mention uniformity of this architecture which is reflected in the fact that convolutional layers are followed by max pooling and use ReLU activation function. Another peculiarity is that feature map dimensions are homogenously reducing by the factor of 2, whilst the depth of the network increases from 64 to 512 levels. Due to such powerful complexity, VGG was able to achieve 25.6% test error on ILSVRC-2012 dataset, although it required to train 138 million weight parameters which demands considerable amount of time and computational resources.

2.4 Transfer learning

It is very uncommon to practically train a whole convolutional network from the very beginning (with arbitrary initialization) because of small number of available datasets of adequate size and vast computational requirements. Instead, usually a ConvNet is pretrained on a huge dataset existing before and, after that it is used either as an extractor of features or an initializer for the given objective. ImageNet is considered as a perfect example of an extremely large dataset, which comprises roughly 1.2 million images classified into 1000 categories [33]. There are multiple possible scenarios of transfer learning, however for the purposes of the project, this thesis focuses only on ConvNet as fixed feature extractor.

Firstly, a ConvNet that was pretrained on large dataset is taken. After removing the last fully-connected layer (outcomes of which are the 1000 class scores for the primary or *source task*), the remaining part of the ConvNet is treated as a fixed feature extractor for another dataset (*target task*). It turns out that the first part of the network pretrained on large dataset is extremely useful and reusable as a preprocessor for the more narrow task. Feature extractor represented by very deep convolutional network usually produces embeddings which are fed into shallower classifier for more narrow task. In this way performance of the pattern recognition is considerably enhanced.

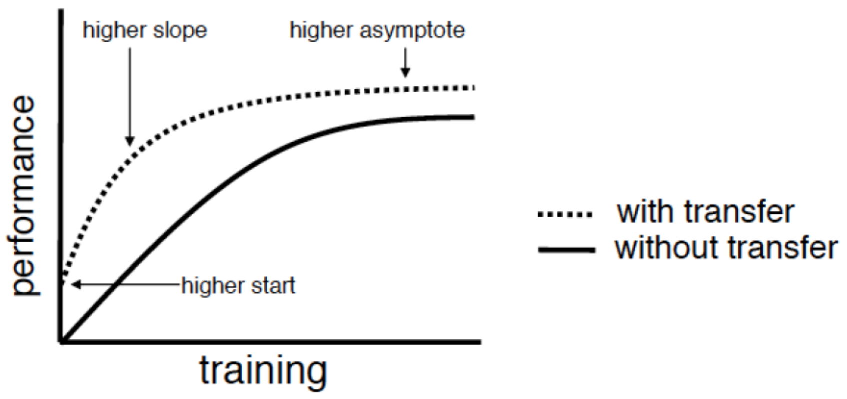


Figure 2.30 Implications of transfer learning [34]

To conclude, application of transfer learning has the following crucial implications:

- *higher start* - improving the quality of training at the initial iterations due to a more careful selection of the initial parameters of the model or some other a priori information.
- *higher slope* - acceleration of convergence of the learning algorithm
- *higher asymptote* - improvement of the upper achievable quality boundary.

2.5 Summary

This chapter provided a thorough review of the theoretical background behind neural networks and machine learning techniques. It started from explanation of trivial perceptron and elaborated on the training process using gradient descent method implemented by backpropagation algorithm. Additionally, various optimization and regularization techniques for improving the performance of training were introduced. Finally, chapter presented convolutional neural networks which are the building blocks for the deep learning architectures. It can be summarized that machine learning techniques can be utilized to approximate decent classifier for recognition of various patterns. Convolutional neural networks have already demonstrated remarkable accuracy in visual classification as well as coped with some music and speech recognition problems. The next chapter focuses on the implementation of DNN for classification of the daily casual sounds (acoustic events that do not include speech or music data and are usually more diverse and chaotic in their structure) and eventual assessment of embedded situationally aware system.

Chapter 3. DESIGN AND IMPLEMENTATION OF THE PLATFORM

Now that all the theoretical background behind audio analysis and pattern classification was described, one can proceed to building a platform for extracting features from environmental sounds using deep convolutional network and mel frequency cepstral coefficients on embedded system. The first step to accomplish this task is to train a deep neural model for classification of features extracted using digital signal processing techniques described above. After obtaining the model one can proceed to transferring the model into the embedded system and testing it in production environment. This chapter provides detailed description and specifications of the project along with methodologies used for accomplishing the goal stated in the introduction of this thesis. Firstly, brief information about the experimental environment is given. Then, two different approaches for training a classifier on top of deep convolutional feature extractor are discussed. Finally, application of trained model in practical embedded system is described.

3.1 Setup of environment for training

In order to build a feature classifier using CNN, one has to carefully design and train problem specific network architecture. Before diving into the world of deep learning, it is desired to gently prepare environment for efficient training process. This includes installing required drivers, preparing datasets and setting up frameworks which are extremely useful for the simplification and abstraction of the low-level concepts. This section briefly introduces experimental and production environment of the project.

3.1.1 Parallel GPU computing

Although neural networks have existed for many years, there are two recent crucial factors which had enormous influence on the widespread utilization of machine learning techniques: tremendous amount of data for training, and effective and powerful parallel computing provided by graphical processing units (GPUs). Application of GPUs enabled training deep neural networks using extremely bigger datasets for learning, in a considerably less time and requiring smaller datacentre infrastructure.

The reason behind incredible acceleration of training speed by GPUs is their parallel computation capabilities. Usually neural networks contain millions of parameters which should undergo same repetitive operations (gradient calculations in forward and backward steps). Architecture of GPU allows to execute these computations simultaneously in a vast number of concurrent threads, which results in a high throughput capacity. In contrast, CPUs are mainly designed for more conventional computing payloads and demonstrates poor performance while training on large datasets. In this project, one of the latest Nvidia GPU graphic cards was used along with CUDA 7.0 hardware acceleration framework. It gave enormous boost to the training which allowed parallel computing and optimized a search for optimal hyperparameters.

3.1.2 Programming frameworks for machine learning

While working on machine learning problems, one has to take care of planning deep learning neural nets architecture and designing efficient calculations of the gradients, weights and activation functions in the form of multidimensional matrixes. Immersing into the low-level computation details facilitates risk of unintended trivial errors and consecutive faults while training or assessment. Machine learning programming

```
b = tf.Variable(tf.zeros([100]))
W = tf.Variable(tf.random_uniform([784,100],-1,1))
x = tf.placeholder(name="x")
relu = tf.nn.relu(tf.matmul(W, x) + b)
C = [...]
s = tf.Session() for step in xrange(0, 10):
          input = ...construct 100-D input array ...
          result = s.run(C, feed_dict={x: input})
          print step, result
```

Figure 3.1 Sample Code snippet from Tensorflow

frameworks reduce the complexity of these operations enabling efficient usage of parallel computing capabilities of GPUs and provide large base of useful predefined functions.

This project is implemented using one of the most widely spread frameworks known as TensorFlow which was built by Google Brain team as a second-generation system for the implementation and deployment of largescale machine learning models [35]. Computations in this framework are defined by a directed graph which consists of several nodes. The graph serves as a representation of dataflow computation, with supplementary functions for enabling some types of nodes to persist and renew their state as well as for looping and branching control statements within the graph. It is constructed by using one of the frontend languages, which TensorFlow supports. For the purpose of this project Python programming language is used. Figure 3.1 illustrates brief code snippet to construct and then run a TensorFlow graph using the Python front end, while corresponding computation chart is depicted on *Figure 3.2*. Distinctive feature of this framework is that it first builds prospective operations flow and then finds most efficient way to execute these commands. Nodes in each graph can have multiple or

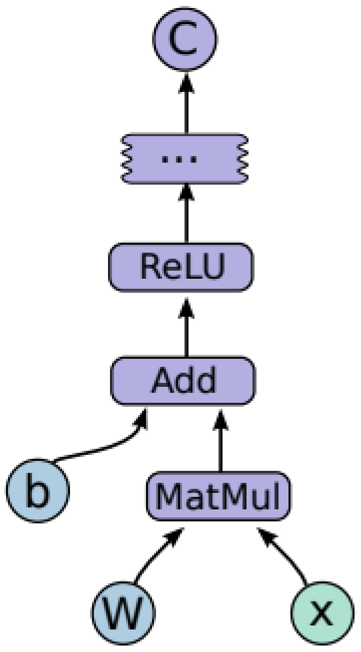


Figure 3.2. Computation graph for code snippet from figure 3.1 [35]

single input, multiple or single output and stand for the initialization of an operation. Values passing through normal edges in the graph (from outputs to inputs) are *tensors*, arbitrary dimensionality arrays where the underlying element type is specified or inferred at graph-construction time [35]. Clients applications reach the TensorFlow core functions by initiating a Session. One of the primary operations provided by Session interface is Run, which accepts a number of output tags that are required to be computed, along with an optional collection of tensors to be supplied into the graph in place of relevant outputs of nodes. Applying the parameters to Run interface, the TensorFlow realization can produce the transitional closure of all nodes that have to be executed for computation of the outputs that were requested and can then organize execution of the corresponding nodes in a sequence that follows their dependencies. Usually, Session is initialized with a graph only once, and then the complete graph or a few separate subgraphs are executed thousands or millions of times via Run calls.

One of the main advantages of the TensorFlow is that after establishment of a computation graph it maps these operations onto the set of available devices. Behind the scenes the framework decides which device (GPU/CPU or distributed implementation of datacenter) to assign the computation for every node in the graph, and then manages the necessary interaction and data flow across device boundaries derived by these placement decisions. This provides a basis for distributed and multi device neural network training which in turn enormously affects the performance.

3.1.3 Production environment

In order to ensure that situationally aware machine is mobile, flexible and unsophisticated for deployment in terms of hardware, this project has taken a focus on implementation of deep convolutional sound classification models deployed on embedded system. One of the most widespread and suitable mobile platforms for accomplishment of this task is Raspberry Pi single-board computer. For the purpose of this project Raspberry Pi model 3 with 1.2 GHz 64-bit quad core processor and on-board communication modules (Wi-Fi, Bluetooth) was utilized. Despite having a decent computational power, Raspberry alone is not capable of making classification prediction using large convolutional model. Therefore, a Movidius Neural Compute Stick produced by Intel was used to optimize operation of deep models produced during training phase. It is compatible with Raspberry Pi, supports TensorFlow programming framework and can run deep architectures like VGGish.

3.2 The datasets

The significance of appropriate dataset and its influence on the performance of the network were discussed in the section 2.2.2, while this section provides brief information about datasets used in the project. Models utilized for feature extraction were trained using AudioSet [36]. It comprises an ontology of 632 sound event classes and a set of 1,789,621 labeled 10-second extractions from YouTube videos. The goal of authors was facilitating research in the field of acoustic event detection and building hierarchically structured dataset which covers a wide range of human and animal sounds, common daily environmental sounds and musical instruments or genres.

Since the aim the project is to build situationally aware system which perceives surrounding environment, more dedicated dataset ESC-50 [37] was used to train classifier networks. It comprises 2 000 labeled environmental sound clips equally aligned between 50 classes (40 recordings per class). For convenience, they are grouped in 5 loosely defined major categories (10 classes per category):

- natural soundscapes and water sounds,
- animal sound
- interior/domestic sounds,
- human (non-speech) sounds,
- exterior/urban noises.

The author of the dataset tried to make acoustic events explicit in the foreground and attenuate background noise when possible, although he states that some recordings may still exhibit overlap of sounds in the background. ESC-50 includes a wide range of sound sources - from very general (cat meowing, laughter, dog barking) to some completely specific (teeth brushing, glass breaking) along with sounds which have tiny differences (airplane and helicopter noise). One of the essential drawbacks of this dataset is the small quantity of recordings available per class. Author explains it by the high cost of manual annotation and extraction, and the decision to establish strong equilibrium between categories despite limited availability of recordings for more uncommon types of sound events. It is noteworthy to mention that the dataset was shuffled into five folds which is extremely handy for separating test and training sets. Particularly, due to small size of dataset, four folds are used for training and remaining one for testing.

3.3 Transfer learning based on VGGish model

The first network to recognize and detect presence of certain environmental sounds in surroundings was trained using Google VGGish model [14] as a feature extractor producing excerpts which are utilized for further processing. The main idea is to pass log mel spectrogram of an input signal through the VGGish model which was pretrained on large AudioSet to extract embeddings, which are then fed as input into additional fully connected layers for classification. All training and assessment operations were accomplished using Nvidia Titan GPU and TensorFlow framework. Subsections below provide detailed description of the architecture utilized in this approach, training process and results obtained after evaluating the model outcomes on test set.

3.3.1 Network architecture

VGGish is a slightly modified version of the VGG model described in section 2.3.5, particularly configuration A which consists of 11 weight layers. Authors altered the input size from 224x224x3 to 96x64 in order to fit the size of log mel spectrogram audio inputs. Further modifications include dropping the final group of convolutional and max pooling layers, which resulted in four groups of conv/pool layers instead of five. Finally, a 1000-wide dense layer at the end was replaced with a 128-wide fully connected layer, which acts as a compact embedding layer. All layers up to and including the final 128-wide embedding layer are defined in the pretrained model provided in the GitHub repository of Google Research team.

VGGish model was trained using audio features extracted from training data by applying techniques described in the section 1.4.2. Particularly all recordings were resampled into 16 kHz mono signal, for which the spectrograms were obtained using magnitudes of the short time Fourier transform applying 25 ms windows every 10 ms of sound clip. These spectrograms were mapped into 64 mel-spaced bins which cover the range of 125-7500 Hz. The value of every bin is transformed after adding a small offset (to avoid numerical issues) by taking a logarithm, which results in a stabilized log mel spectrogram patches of 96 × 64 bins. These patches form the input to VGGish model. A softmax classifier function was used to evaluate the cost of training, as the labels are mutually exclusive but each of them can have a certain portion in the input.

Embeddings generated from the VGGish model were used to train a classifier network consisting of 2 fully connected layers with 1024 and 50 nodes respectively. Figure 3.3 illustrates simplified computation graph which was generated while training this network. Entire representation of computation graph is given in Appendix 2.

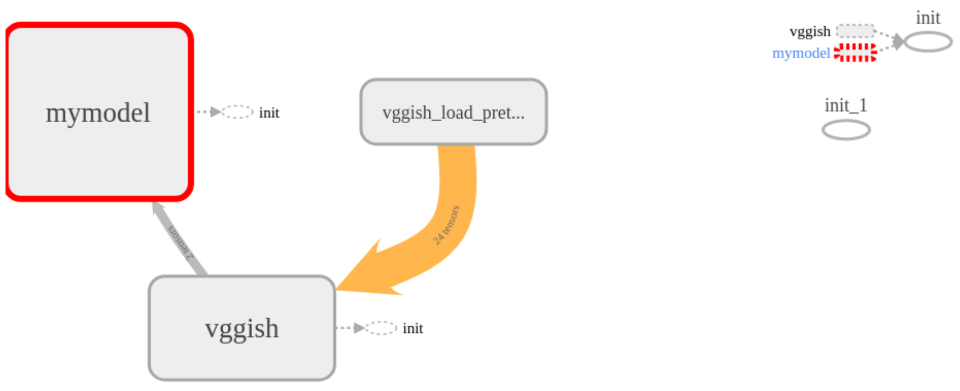


Figure 3.3. Computation graph generated by TensorFlow framework

Clearly, the pretrained checkpoint is used to initialize VGGish model which extracts the required embeddings and applies them to the model defined by our approach. Training process and corresponding performance results are presented in chapter 4.

3.4 Knowledge transfer from weakly labelled audio

Because of the unsuccessful outcomes of the first approach, alternative method was utilized to increase the efficiency of the recognition which is crucial for practical implementation. Similarly, deep convolutional network was used to efficiently transfer knowledge from weakly labelled audio data, however the architecture and methodology were different [38]. The model in this approach is designed by using numerous different structures to efficiently transfer learned representations from a CNN based sound event classifier trained on a large-scale training data (AudioSet). The proposed network architecture for weak label learning executes smoothly and productively on audio clips of various length which makes it suitable for transductive transfer learning also known as domain adaptation. This approach implies application of different datasets for learning representations and further training of a classifier. More detailed information and specifications are discussed below.

3.4.1 Network architecture

The architecture of the feature extractor network proposed for this method is presented on Figure 3.4. Layers B1-B5 are comprised of two convolutional layers, each of which passes the output through batch normalization followed by ReLU activation function, and max pooling in the end. Layer B6 is made up of one convolutional layer followed

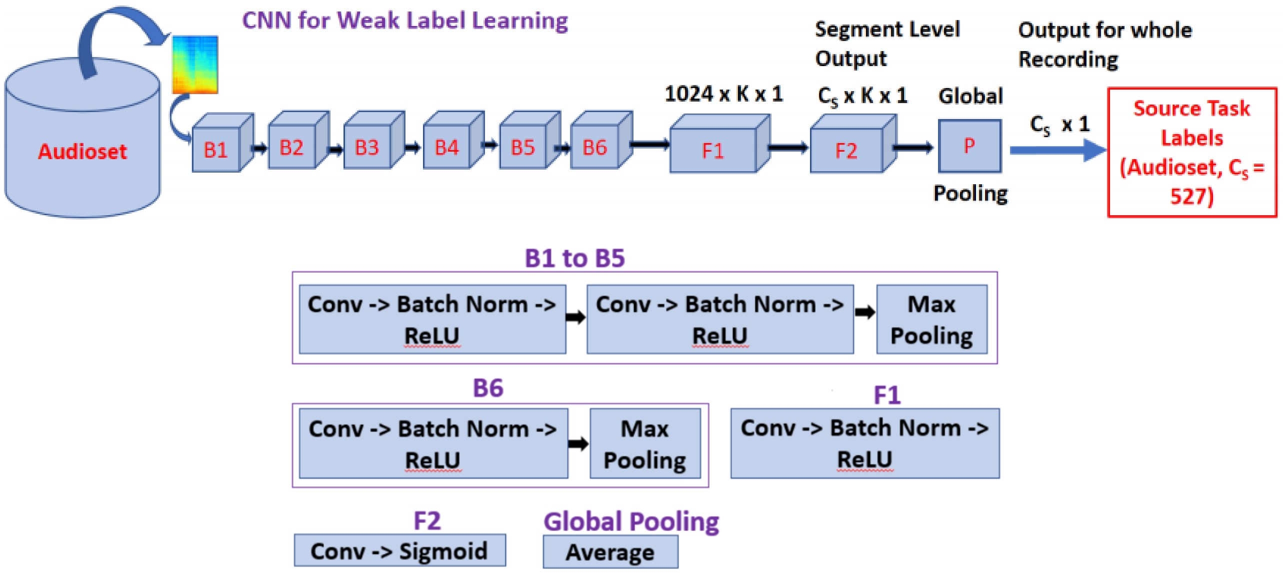

CNN for Weak Label Learning
Audioset
B1
B2
B3
B4
B5
B6
1024 x K x 1
F1
Segment Level Output
C_S x K x 1
F2
Global
P
Pooling
Output for whole Recording
C_S x 1
Source Task Labels (Audioset, C_S = 527)
B1 to B5
Conv -> Batch Norm -> ReLU
Conv -> Batch Norm -> ReLU
Max Pooling
B6
Conv -> Batch Norm -> ReLU
Max Pooling
F1
Conv -> Batch Norm -> ReLU
F2
Conv -> Sigmoid
Global Pooling
Average

Figure 3.4. Architecture of feature extractor trained on the AudioSet [38]

by a max pooling. All the blocks B1-B6 use 3x3 filter size, while the number of filters doubles in each layer starting from 16 in B1 and culminating in 512 kernels in B6. F1 is also a convolutional layer with ReLU activation consisting of 1024 kernels of size 2x2 used with a stride of 1 and no padding. F2 is the secondary convolutional output layer comprising C_S kernels with size of 1x1 and sigmoid output. This block generates segmented output (C_S x K x 1), where K is the number of segments and Cs is the number of classes (in source task). Global pooling layer is used to aggregate the segment level outputs and produces Cs×1-dimensional output for the whole audio clip.

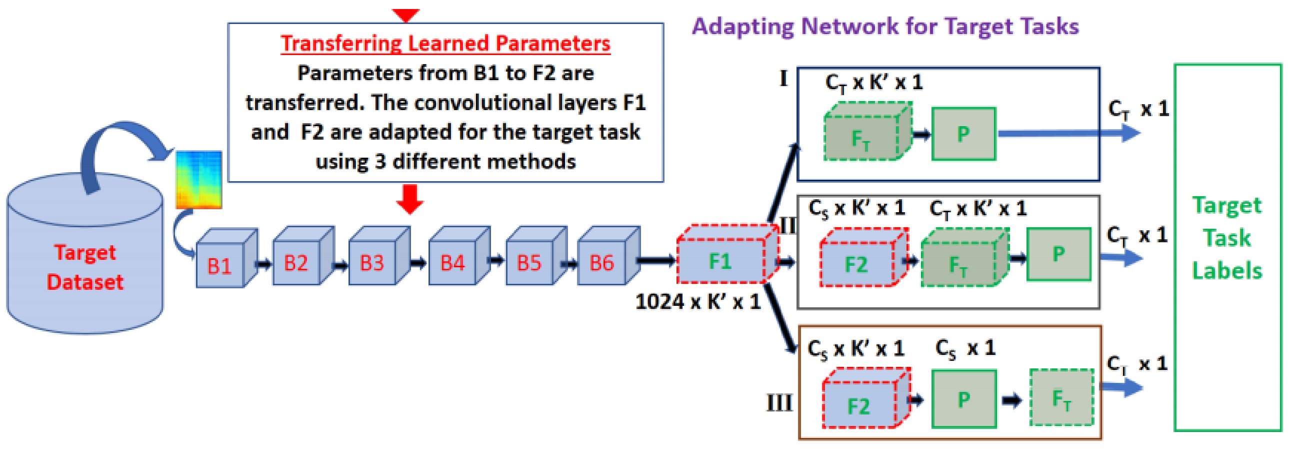

Transferring Learned Parameters
Parameters from B1 to F2 are transferred. The convolutional layers F1 and F2 are adapted for the target task using 3 different methods
Adapting Network for Target Tasks
Target Dataset
B1
B2
B3
B4
B5
B6
F1
1024 x K' x 1
I
C_T x K' x 1
F_T
P
C_T x 1
II
C_S x K' x 1
C_T x K' x 1
F2
F_T
P
C_T x 1
III
C_S x K' x 1
C_S x 1
F2
P
F_T
C_T x 1
Target Task Labels

Figure 3.5 Transferring knowledge into shallow classifier network by passing target dataset (ESC-50) through the whole network [38]

The feature extractor network trained on the AudioSet is applied to produce learning representations for recordings in the target task. Figure 3.5 demonstrates this procedure. Embeddings from layer F1 (1024×K ×1) and F2 (C_S ×K ×1) serve as base representations for audios which are then mapped to full audio level representations using either max or average pooling. Eventually, 1024 (F1) or C_S (F2) dimensional embeddings are obtained for full sound clips. During training network using target dataset ESC-50, the layers from B1 to B6 were used to embed knowledge from source audio data into F1, which is then used to train classifiers for target task. Figure 3.5 illustrates 3 different methods to accomplish this task, each of which used dropout regularization afterwards. First method uses direct adaptation to the target dataset by replacing F2 with F_T producing C_T x K x 1 dimensional output which is passed through a global pooling. Second approach adds new convolutional layer $F_{T\ only}$ after block F2 which allows to capture target characteristics after translating embeddings into source label F2 and then move to target label space. Final approach implies positioning F_T after the output of the global pooling. In all three cases, only the blocks indicated with dashed edges are trainable, while remaining layer weights are kept constant. It is noteworthy to mention that final embedding layer in classifier network was comprised of 512 units instead of 527 classes, which considerably improved the performance. In all configurations softmax cross entropy cost function used to estimate probability distributions over the whole range of classes.

3.5 Summary

This chapter introduced the methodology, hardware and software solutions utilized to implement the project. This includes the detailed information about the neural network programming frameworks, GPU hardware specialized for training, embedded systems in the form of Raspberry PI equipped with Movidius Neural Computation Stick and two different architecture approaches. The next chapter focuses on the training process and the provides discussion of results obtained during training and testing the architectures described above.

Chapter 4. RESULTS AND DISCUSSION

This chapter presents the training and testing results obtained from two model architectures described in previous chapter. It also provides the testing outcomes of the trained sound classifier on the embedded system in the form of Raspberry Pi which eventually completes the situationally-aware mobile system.

4.1 Training and test results on VGGish model

The results of training the network with the batch size of 256 examples and learning rate of 0.01 for 230 epochs are given in the Table 5. Although the training set accuracy reaches considerable value of almost 90%, this model completely fails on the test set with poor ~40% accuracy. Cost value shows that there is still a room for improvement however it turns out that the model is drastically overfits the data.

Table 5. VGGish evaluation results

Training set accuracy	**Test set accuracy**	**Average cost**
0.899194	0.3915	0.73511

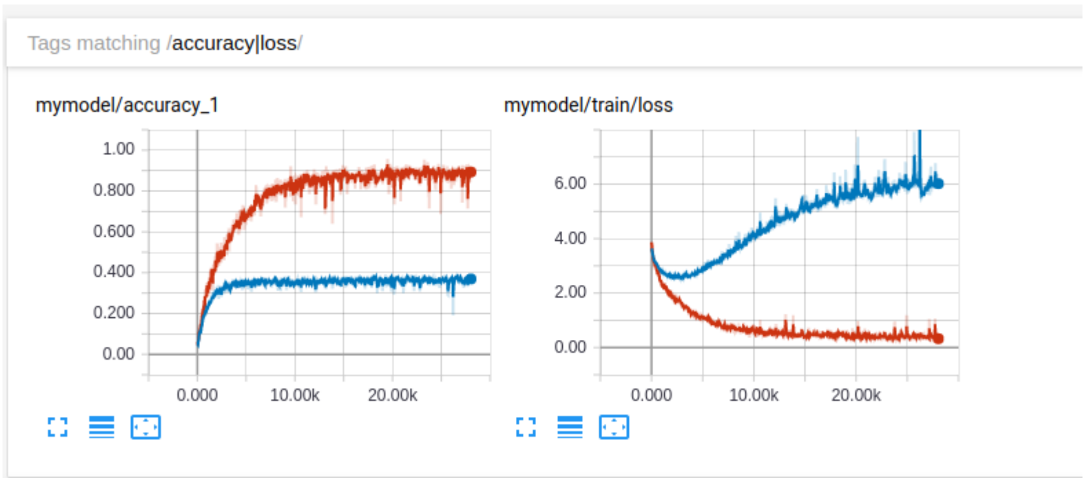
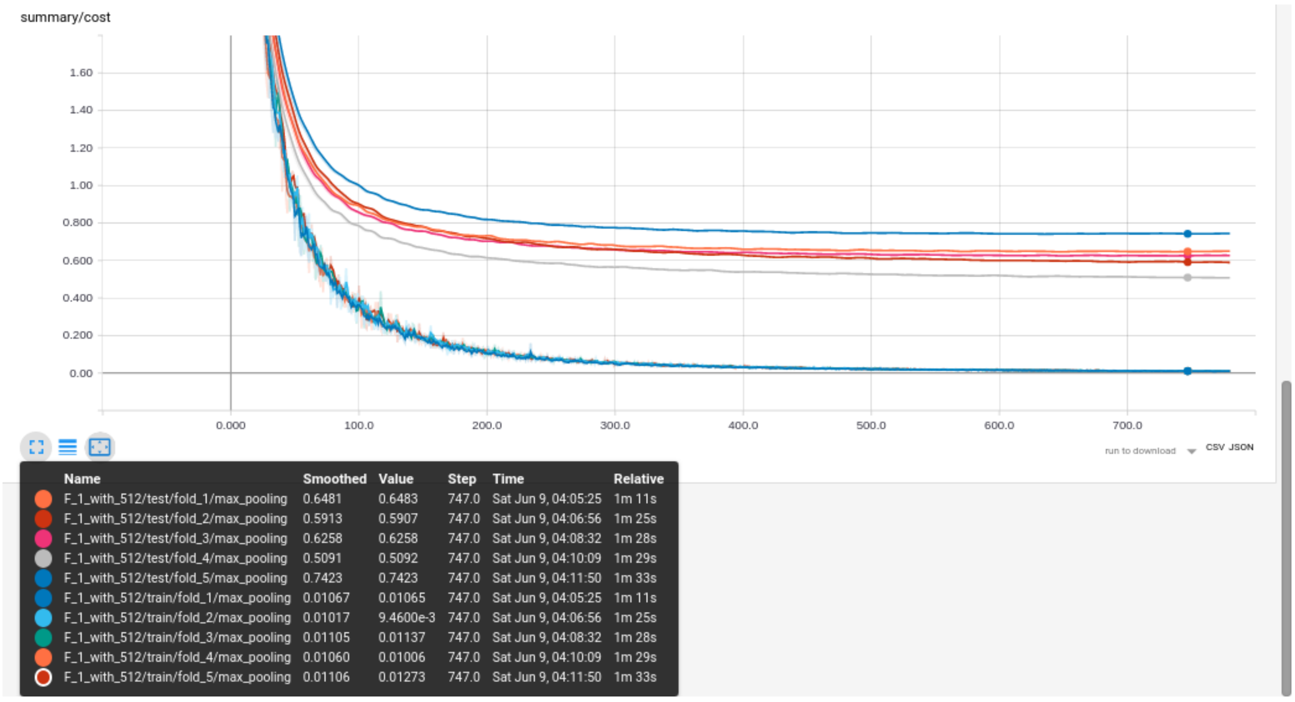


Figure 4.1 Cost and accuracy of training classifier on top of VGGish model. Blue graph stands for test set, while red represents a training set

Figure 4.1 provides straightforward evidence of poor training and enormous overfitting which can be concluded from the large gap between training and test set accuracy/cost. Applying regularization techniques and training the network using alternative configurations did not improve the test set results. There can be multiple reasons behind failure of this model including overcomplexity of the given architecture as well as dimensionality mismatch issues. One possible reason could be ambiguity of the learned representations in the feature extractor and mismatch with the target task. Eventually, after a lot of unlucky attempts, it was decided to switch to alternative architecture.

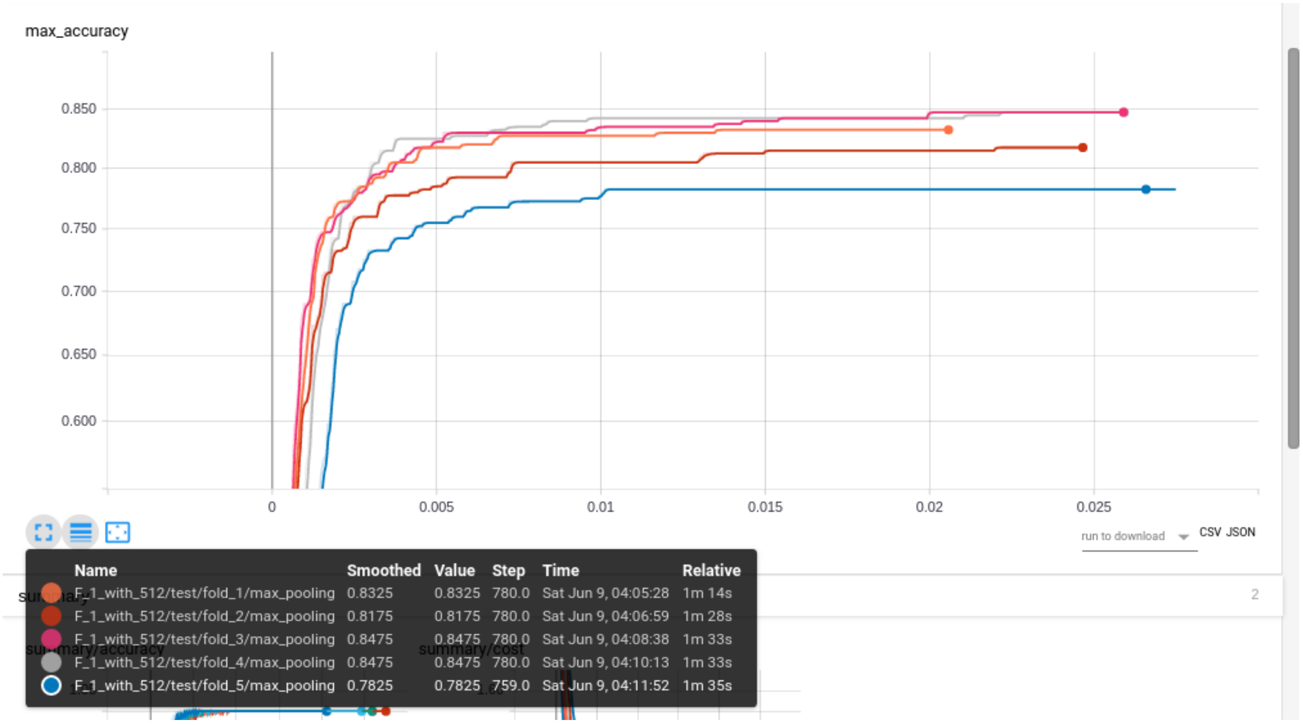
4.2 Training and test results of weak feature extractor model

Before training a shallow classifier, entire dataset was passed through feature extractor containing all layers up to F1 on Figure 3.4, which produced 1024x5x1 embeddings for each example totalling into 1024x5x2000 dimensional vector. This embeddings vector is then transposed in order to meet input requirements of TensorFlow. The best results were achieved while using third configuration from previous chapter. Figure 4.2 and Figure 4.3 summarize the visualization of the network training process. Clearly,



	Name	Smoothed	Value	Step	Time	Relative
●	F_1_with_512/test/fold_1/max_pooling	0.6481	0.6483	747.0	Sat Jun 9, 04:05:25	1m 11s
●	F_1_with_512/test/fold_2/max_pooling	0.5913	0.5907	747.0	Sat Jun 9, 04:06:56	1m 25s
●	F_1_with_512/test/fold_3/max_pooling	0.6258	0.6258	747.0	Sat Jun 9, 04:08:32	1m 28s
●	F_1_with_512/test/fold_4/max_pooling	0.5091	0.5092	747.0	Sat Jun 9, 04:10:09	1m 29s
●	F_1_with_512/test/fold_5/max_pooling	0.7423	0.7423	747.0	Sat Jun 9, 04:11:50	1m 33s
●	F_1_with_512/train/fold_1/max_pooling	0.01067	0.01065	747.0	Sat Jun 9, 04:05:25	1m 11s
●	F_1_with_512/train/fold_2/max_pooling	0.01017	9.4600e-3	747.0	Sat Jun 9, 04:06:56	1m 25s
●	F_1_with_512/train/fold_3/max_pooling	0.01105	0.01137	747.0	Sat Jun 9, 04:08:32	1m 28s
●	F_1_with_512/train/fold_4/max_pooling	0.01060	0.01006	747.0	Sat Jun 9, 04:10:09	1m 29s
●	F_1_with_512/train/fold_5/max_pooling	0.01106	0.01273	747.0	Sat Jun 9, 04:11:50	1m 33s

Figure 4.2. Training and test set cost value graphs across 5 different folds



	Name	Smoothed	Value	Step	Time	Relative
●	F_1_with_512/test/fold_1/max_pooling	0.8325	0.8325	780.0	Sat Jun 9, 04:05:28	1m 14s
●	F_1_with_512/test/fold_2/max_pooling	0.8175	0.8175	780.0	Sat Jun 9, 04:06:59	1m 28s
●	F_1_with_512/test/fold_3/max_pooling	0.8475	0.8475	780.0	Sat Jun 9, 04:08:38	1m 33s
●	F_1_with_512/test/fold_4/max_pooling	0.8475	0.8475	780.0	Sat Jun 9, 04:10:13	1m 33s
●	F_1_with_512/test/fold_5/max_pooling	0.7825	0.7825	759.0	Sat Jun 9, 04:11:52	1m 35s

Figure 4.3. Accuracy graphs for 5 different folds of ESC-50 used as a test set

Table 6. Average test set accuracy of knowledge transfer model

	Test fold 1	**Test fold 2**	**Test fold 3**	**Test fold 4**	**Test fold 5**
Accuracy	83.25%	81.75%	84.75%	84.75%	78.25%
Average	**82.55%**				

the network performed more than two times better than previous model, however there is still a gap between the loss function and accuracy curves which informs about slight overfitting. Since the data in the different folds is not uniform performance of the network varies with changing the number of test fold. Table 6 demonstrates average accuracy of the network across different test folds which are presented in ESC-50. It is noteworthy to mention that this model outperformed the accuracy of the humans (81.3%). One can conclude that the training checkpoint can be efficiently used in order to recognize audio events in required applications.

4.3 Testing the model on embedded system

The trained model was successfully transferred into Raspberry Pi embedded system running on Raspbian Linux based OS. In order to add computational performance and effectively tackle with neural computations, Movidius chipset was used as an extension of Raspberry. The sound recordings were made using high definition microphone with integrated noise attenuating filters. It was used to capture the input signal with the interval of 3 seconds which are then passed into feature extractor network. Embeddings from this network are successfully fed into trained classifier which effectively recognizes the environmental sounds and detects events, providing the confidence probabilities. This process in conducted in the infinite loop for continuous measurement, processing and classification of the input from surrounding environment and completes situationally-aware system ready to detect and response to certain events occurring around.

Conclusions

To conclude, this thesis has described a procedure to build effective situationally aware embedded system which uses environmental sound classification for getting insights into the surrounding context. The overall performance of the deep learning classifier outperformed the accuracy level of human which can be considered as successful accomplishment of the initial goal of the project. Using the approach presented in the third chapter, one can train neural network on larger datasets and make the model more generalized and prune to the details. With the reduction of size of embedded computational systems, situation aware machines can emerge in vast amount of applications including safety, security and tracking of global ecological changes. They can be modified to detect and inform about certain events through the network and thus prevent crimes, establish a basis for ecology protection systems and track the global patterns such as climate change. Future work can combine auditory information with visual data in order to produce full situational awareness and resemble human perception. Rise in the performance and mobility of the embedded systems along with more efficient deep network architectures can shift machine perception in a whole new level.

Bibliography

[1] R. Want, A. Hopper, V. Falcao and J. Gibbons, “The Active Badge Location System,” *ACM Trans. Inf. Syst.*, vol. 10, no. 1, pp. 91-102, 1992.

[2] B. Schilit, N. Adams and R. Want, “Context-Aware Computing Applications,” in *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, Washington, 1994.

[3] A. Barghi, A. R. Kosari, M. Shokri and S. Sheikhaei, “ Intelligent lighting control with LEDS for smart home,” in *Smart Grid Conference, 10.1109/SGC.2014.7090861*, 2014.

[4] L. Niklasson, M. Riveiro, F. Johansson, A. Dahlbom, G. Falkman, T. Ziemke, C. Brax, T. Kronhamn, M. Smedberg, H. Warston and P. Gustavsson, “A Unified Situation Analysis Model for Human and Machine Situation Awareness,” in *INFORMATIK 2007: Informatik trifft Logistik.*, 2007.

[5] M. R. Endsley, “Toward a Theory of Situation Awareness in Dynamic Systems.,” *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 37, no. 1, pp. 32-64, 1995.

[6] B. C. Pijanowski, L. J. Villanueva-Rivera, S. L. Dumyahn, A. Farina, B. L. Krause, B. M. Napoletano, S. H. Gage and N. Pieretti, “Soundscape Ecology: The Science of Sound in the Landscape,” *BioScience*, vol. 61, no. 3, pp. 203-216, 2011.

[7] J. Dzieza, “Scientists are recording the sound of the whole planet,” The Verge, 28 August 2014. [Online]. Available: https://www.theverge.com/2014/8/28/6071399/scientists-are-recording-the-sound-of-the-whole-planet. [Accessed 30 May 2018].

[8] A. V. Openheim, R. W. Schafer and J. R. Buck, Discrete-time signal processing (2nd ed), New Jersey: Prentice Hall, 1999.

[9] X. Huang, A. Acero and H.-W. Hon, Spoken language processing: a guide to theory, algorithm, an system development, Prentice Hall, 2001.

[10] W. C. J. and W. T. J., “An algorithm for the machine calculation of complex Fourier,” *Math. Comput*, vol. 19, pp. 297-301, 1965.

[11] L. Ma, D. Smith and B. Milner, “Context Awareness using Environmental Noise Classification,” in *Mařík V., Retschitzegger W., Štěpánková O. (eds) Database and Expert Systems Applications.*, 2003.

[12] K. Prahallad, *Speech Technology: A Practical Introduction. Lecture notes on Spectrogram, Cepstrum and Mel-Frequency Analysis*, Carnegie Mellon University, International Institute of Information Technology Hyderabad.

[13] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, Lake Tahoe, Nevada, 2012.

[14] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. Weiss and K. Wilson, *CNN Architectures for Large-Scale Audio Classification*, arXiv:1609.09430, 2017.

[15] McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, pp. 115-133, 1943.

[16] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain," *Psychological Review,* pp. 65-386, 1958.

[17] "CS231n: Convolutional Neural Networks for Visual Recognition," Stanford university, 2016. [Online]. Available: http://cs231n.github.io/. [Accessed 05 2018].

[18] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[19] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning.," in *From Natural to Artificial Neural Computation*, Springer, Berlin, Heidelberg, 1995.

[20] Y. LeCun, L. Bottou, G. Orr and K. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade, this book is an outgrowth of a 1996 NIPS workshop*, Berlin,Heidelberg, Springer, 1998, pp. 9-50.

[21] S. Russel and P. Norvig, Artificial Intelligence: A Modern Approach, Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[22] A. Ng, "Neural networks and deep learning," DeepLearning.ai.

[23] S. Ruder, *An overview of gradient descent optimization algorithms,* arXiv:1609.04747v2 [cs.LG], 2017.

[24] B. D. Ripley, Pattern Recognition and Neural Networks, Cambridge Universtity Press, 1996.

[25] S. Raschka, Python Machine Learning, Packt Publishing; 1st edition , 2015.

[26] C. M. Bishop, Neural Networks for Pattern Recognition, New York: Oxford University Press, 1995.

[27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research,* vol. 15, no. 1, pp. 1929-1958, 2014.

[28] G. Hinton, N. Srivastava and K. Swersky, *Overview of mini-batch gradient descent. Lecture notes,* Coursera, 2012.

[29] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278-2324, 1998.

[30] LISA lab, "Convolution arithmetic tutorial," Theano, 21 November 2017. [Online]. Available: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html. [Accessed 2 June 2018].

[31] "Student Notes: Convolutional Neural Networks (CNN) Introduction," Belajar Pembelajaran Mesin, [Online]. Available: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/. [Accessed 05 June 2018].

[32] A. Zisserman and K. Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *arXiv:1409.1556v6 [cs.CV]*, 2015.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, *ImageNet: A Large-Scale Hierarchical Image Database,* CVPR09, 2009.

[34] L. Torrey and J. Shavlik, "Transfer Learning," *Handbook of Research on Machine Learning Applications,* 2009.

[35] M. Abadi and e. al., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,* Google Research, 2015.

[36] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, W. L. Aren Jansen, R. C. Moore, M. Plakal and M. Ritter, "Audio Set: An ontology and human-labeled dataset for audio events," in *Proc. IEEE ICASSP 2017,* New Orleans, 2017.

[37] K. Piczak, "ESC: Dataset for Environmental Sound Classification.," in *Proceedings of the 23rd ACM international conference on Multimedia,* 2015.

[38] A. Kumar, M. Khadkevich and C. Fugen, *Knowledge Transfer from Weakly Labeled Audio using Convolutional Neural Network for Sound Events and Scenes,* arXiv:1711.01369 [cs.SD], 2017.

[39] G. Chen and D. Kotz, "A Survey of Context-Aware Mobile Computing Research," Dartmouth College, Hanover, NH, USA, 2000.

[40] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith and P. Steggles, "Towards a Better Understanding of Context and Context-Awareness," in *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing,* Karlsruhe, Germany, 1999.

[41] T. Mullet, *Predicting the Distribution of Human-made Noise in the Soundscape: An Indicator of Habitat Quality,* University of Biology and Wildlife.

[42] V. Genin, "Optimizations of Gradient Descent," Datascience deep dive, 25 March 2016. [Online]. Available: http://dsdeepdive.blogspot.com/2016/03/optimizations-of-gradient-descent.html. [Accessed 1 June 2016].

Appendix A

Pseudo code for Tukey-Cooley FFT algorithm

```
$X_{0,\ldots,N-1}$ ← **ditfft2**($x$, $N$, $s$):              *DFT of* $(x_0, x_s, x_{2s}, \ldots, x_{(N-1)s})$:
    if $N = 1$ then
        $X_0 \leftarrow x_0$                                 *trivial size-1 DFT base case*
    else
        $X_{0,\ldots,N/2-1}$ ← **ditfft2**($x$, $N/2$, $2s$)          *DFT of* $(x_0, x_{2s}, x_{4s}, \ldots)$
        $X_{N/2,\ldots,N-1}$ ← **ditfft2**($x$+s, $N/2$, $2s$)         *DFT of* $(x_s, x_{s+2s}, x_{s+4s}, \ldots)$
        for $k = 0$ to $N/2-1$                           *combine DFTs of two halves into full DFT:*
            $t \leftarrow X_k$
            $X_k \leftarrow t + \exp(-2\pi i\, k/N)\, X_{k+N/2}$
            $X_{k+N/2} \leftarrow t - \exp(-2\pi i\, k/N)\, X_{k+N/2}$
        endfor
    endif
```

Here, ditfft2(x,N,1), computes X=DFT(x) out-of-place by a radix-2 DIT FFT, where N is an integer power of 2 and s=1 is the stride of the input x array. x+s denotes the array starting with xs.

Cepstral Analysis

$$X[k] = H[k]E[k]$$
$$\big||X[k]|\big| = \big||H[k]|\big|\ \big||E[k]|\big|$$
$$\big||\quad|\big| - denotes\ magnitude$$

Take a logarithm on both sides:

$$log\big||X[k]|\big| = log\big||H[k]|\big| + log\big||E[k]|\big|$$

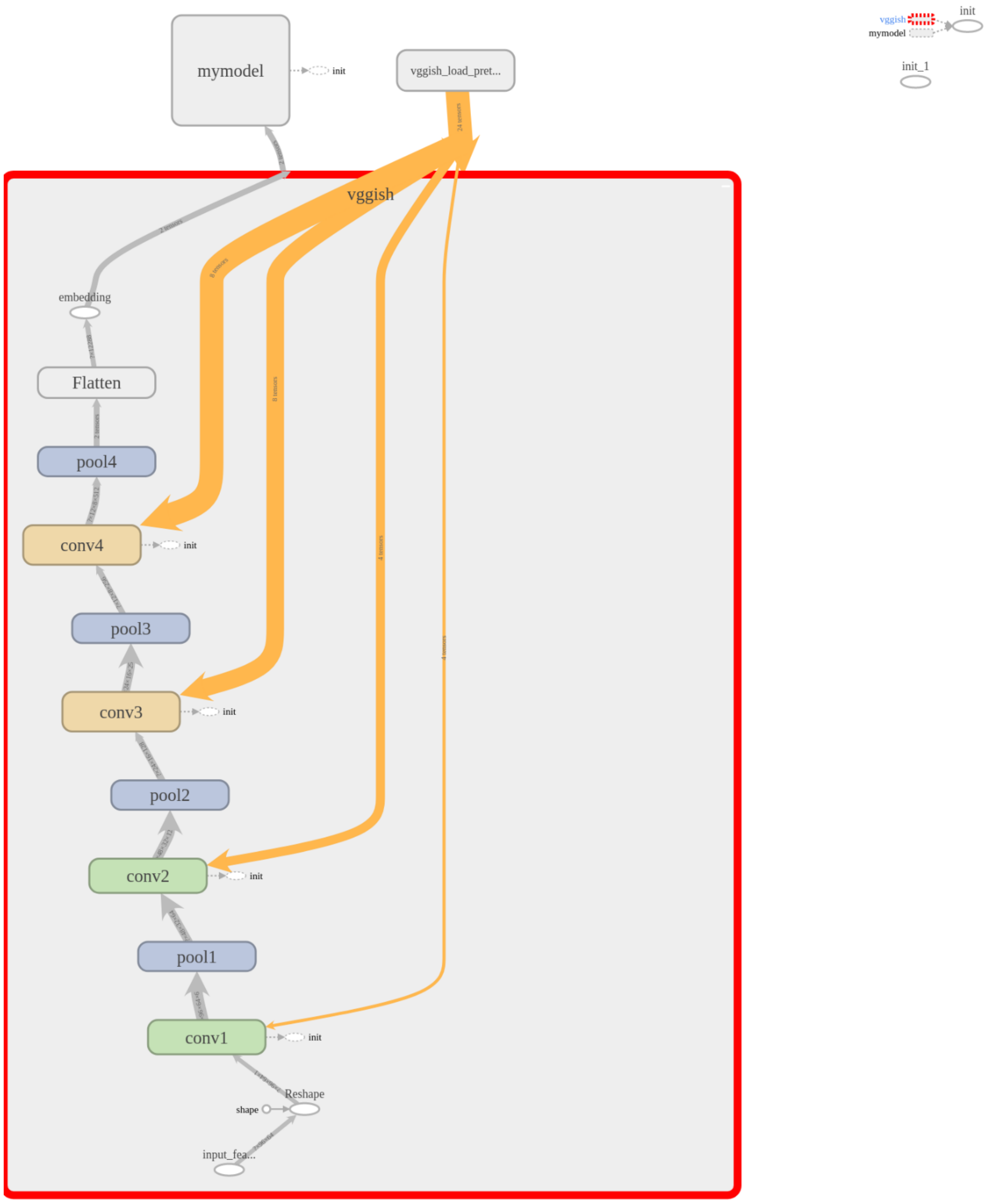
Take inverse FFT on both sides:

$$x[k] = h[k] + e[k]$$

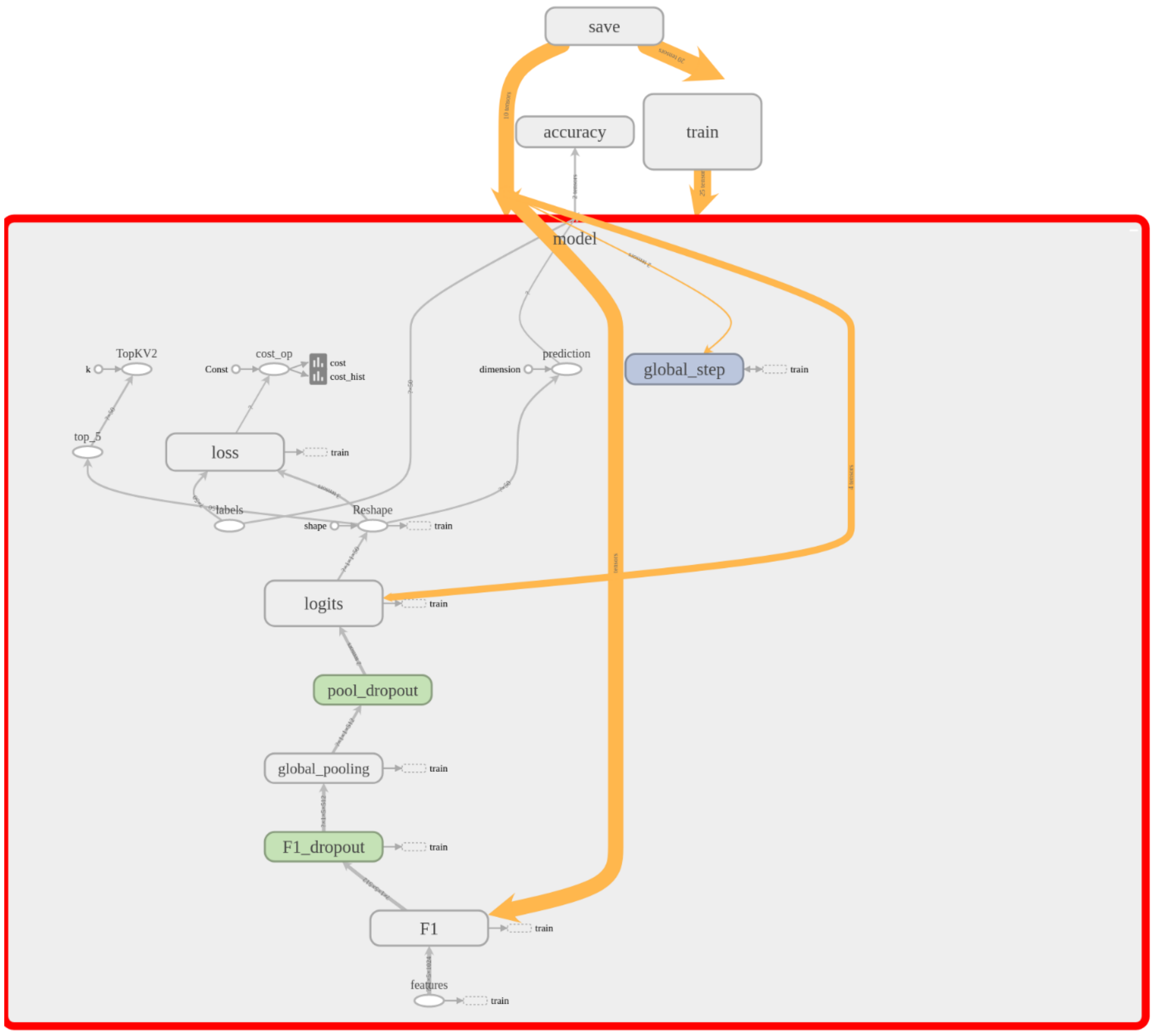
Here $x[k]$ is referred to as Cepstrum (inverse FFT of spectrum $X[k]$), $e[k]$ stands for spectral details, $h[k]$ is obtained by considering the low frequency region of x[k] and represents the spectral envelope which is widely used as feature for speech recognition [12].

Appendix B

Computation graph of VGGish model



Classifier network on top of weakly labeled feature extractor



save
accuracy
train
model
TopKV2
k
cost_op
Const
cost
cost_hist
prediction
dimension
global_step
train
top_5
loss
train
labels
Reshape
shape
train
logits
train
pool_dropout
global_pooling
train
F1_dropout
train
F1
train
features
train