DEEP LEARNING APPROACH FOR LABORATORY MICE GRIMACE SCALING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

MUSTAFA ERAL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

OCTOBER 2016
DEEP LEARNING APPROACH FOR LABORATORY MICE GRIMACE SCALING

submitted by MUSTAFA ERAL in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University by,

Prof. Dr. Gülbin Dural Ünver  Dean, Graduate School of Natural and Applied Sciences
Prof. Dr. Tolga Çiloğlu  Head of Department, Electrical and Electronics Engineering
Prof. Dr. Uğur Halıcı  Supervisor, Electrical and Electronics Engineering Department, METU

Examing Committee Members:
Prof. Dr. Gözde Bozdağlı Akar  Electrical and Electronics Engineering Department, METU
Prof. Dr. Uğur Halıcı  Electrical and Electronics Engineering Department, METU
Assoc. Prof. Dr. İlkay Ulusoy  Electrical and Electronics Engineering Department, METU
Assoc. Prof. Dr. Emine Eren Koçak  Institute of Neurological Sciences and Psychiatry, Hacettepe U.
Assist. Prof. Dr. Elif Vural  Electrical and Electronics Engineering Department, METU

Date:
I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: MUSTAFA ERAL

Signature :
Deep learning is extremely attractive research topic in pattern recognition and machine learning areas. Applications in speech recognition, natural language processing, and machine vision fields gained huge acceleration in performance by employing deep learning. In this thesis, deep learning is used for medical purposes in order to scale pain degree of drug stimulated mice by examining facial grimace. For this purpose each frame in the videos in the training set were scaled manually by experts according to Mouse Grimace Scaling (MGS) manual and these frames were used for training a convolutional neural network. For testing the network, another set of videos which was not used for training before, was used. In order to show the classification power of convolutional neural networks, the same classification tasks are performed with some classic kernel based machine learning algorithms and results are compared. For training and testing, a workstation having two powerful graphic card (GPU) is used.
ÖZ

LABORATUVAR FARESİ YÜZ AĞRI İFADESİ DERECELendirMesi İÇIN DERİN ÖĞRENME

Eral, Mustafa
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Uğur Halıcı

Ekim 2016, 76 sayfa

Anahtar Kelimeler: Sinir Ağları, Derin Öğrenme, Evrişimli Ağlar, Makine Öğrenmesi, Farelerde Ağrı Derecelendirme
To my brothers..
ACKNOWLEDGMENTS

I would like to thank my supervisor Professor Uğur Halıcı for her constant support, guidance and friendship. It was a great honor to work with her. I also would like to thank METU Computer Vision And Intelligent Systems Research Lab members and Hacettepe University Institute of Neurological Sciences and Psychiatry animal lab scientists for their work on creating the database. I also thank to my company, Aselsan, for providing me free time to maintain my master education.

I own Mert Çokluk a debt of gratitude for reviewing all of the chapters of the thesis. I also own Sümeyye Eral, Mustafa Bilgiç, Sanem Elbaşı, Salih Mansur, and Başak Ağin Dönmez, gratitudes for reviewing some chapters of this work.

This study is partially supported under TUBITAK 11SE248-Automatic Evaluation of Pain Related Facial Expressions in Mice (Mice Mimic) Project.
# TABLE OF CONTENTS

ABSTRACT ............................................................... v

ÖZ ................................................................. vii

ACKNOWLEDGMENTS ...................................................... x

TABLE OF CONTENTS ..................................................... xi

LIST OF TABLES ............................................................ xv

LIST OF FIGURES .......................................................... xvi

CHAPTERS

1 INTRODUCTION ............................................................. 1

1.1 Motivation .......................................................... 1

1.2 Contribution ......................................................... 1

1.3 Organization of the Thesis ............................................ 2

2 LITERATURE SURVEY ................................................... 5

3 BACKGROUND INFORMATION ......................................... 9

3.1 Artificial Neural Networks .......................................... 9

3.1.1 Introduction ..................................................... 9

3.1.2 Artificial Neuron ................................................ 9
3.1.3 Multi-Layer Neural Networks .......................... 10
3.1.4 Neuron Output Functions .............................. 11
  3.1.4.1 Perceptron .................................. 11
  3.1.4.2 Sigmoid Function .............................. 11
  3.1.4.3 ReLU ........................................ 12
3.1.5 Cost Functions ........................................ 12
  3.1.5.1 Quadratic Cost Function ......................... 13
  3.1.5.2 Cross Entropy Cost Function ..................... 13
  3.1.5.3 Soft-Max Regression ............................ 14
3.2 Deep Learning ............................................ 15
  3.2.1 Gradient Descent ................................... 15
    3.2.1.1 Stochastic Gradient Descent .................... 16
  3.2.2 Backpropagation Algorithm ......................... 17
    3.2.2.1 Main Idea .................................. 17
    3.2.2.2 Notation .................................... 17
    3.2.2.3 Derivation .................................. 18
    3.2.2.4 The Algorithm ............................... 20
  3.2.3 Convolutional Neural Networks ...................... 21
    3.2.3.1 Motivation .................................. 21
    3.2.3.2 Architecture ............................... 21
    3.2.3.3 Backpropagation in Convolutional Layers ......... 24
3.2.3.4 Backpropagation in Max-Pooling Layers 24

3.2.4 Techniques to Improve Training Performance . . 24
  3.2.4.1 Dropout . . . . . . . . . . . . . . . . 26
  3.2.4.2 Using Adam Optimizer . . . . . . . 26

3.3 Kernel Methods . . . . . . . . . . . . . . . . . . . . . . . . 27
  3.3.1 Support Vector Machines . . . . . . . . . . . . . . 28
  3.3.2 Formulation . . . . . . . . . . . . . . . . . . . . 28
  3.3.3 Principal Component Analysis . . . . . . . . . . . 29
    3.3.3.1 Formulation . . . . . . . . . . . . . 30

3.4 K-Nearest Neighbors Algorithm . . . . . . . . . . . . . . . 31

4 THE MOUSE GRIMACE DATA-SET . . . . . . . . . . . . . . . . 33
  4.1 Usage . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

5 THE MICE-MIMIC-NET . . . . . . . . . . . . . . . . . . . . . . . . 39
  5.1 Network Architecture . . . . . . . . . . . . . . . . . . . . 39
    5.1.1 Input Layer . . . . . . . . . . . . . . . . . . . . 39
    5.1.2 First Convolutional and Max Pooling Layer . . . 40
    5.1.3 Second Convolutional and Max Pooling Layer . . . 41
    5.1.4 Fully-Connected Layer . . . . . . . . . . . . . . 41
    5.1.5 The Soft-Max Output Layer . . . . . . . . . . . 42
  5.2 Why This Network . . . . . . . . . . . . . . . . . . . . . . 42

6 EXPERIMENTAL WORK . . . . . . . . . . . . . . . . . . . . . . . 45

xiii
6.1 Choosing Network Parameters ........................................... 45
  6.1.1 Adam Optimizer Learning Rate ............................... 45
  6.1.2 Number of Convolutional Layers ............................ 45
  6.1.3 Kernel Size ......................................................... 46
  6.1.4 Number of Filters ............................................... 47
  6.1.5 Number of Fully Connected Layers ........................ 48
  6.1.6 Dropout Probability ............................................. 48

6.2 Training With Single Facial Elements .............................. 49
  6.2.1 Data Augmentation ............................................. 50

6.3 The Final Network ..................................................... 51

6.4 Benchmarking With Classical Methods ............................. 53
  6.4.1 Multilayer Perceptron ......................................... 53
  6.4.2 Classifying With SVMs ......................................... 56
  6.4.3 Classifying With K Neighbors Classifier ................. 57

6.5 Benchmarking All Methods .......................................... 57
  6.5.1 Receiver Operating Characteristics (ROC) Comparisons ................................. 59

6.6 Representivity of the Data Set ..................................... 60
  6.6.1 Using A Completely Different Video As Test Data 62

7 CONCLUSIONS ............................................................ 67

REFERENCES ................................................................. 69

Glossary ................................................................. 76
LIST OF TABLES

TABLES

Table 4.1 Each item of the label array in the data-set holds distinct features about corresponding face in the database. ... 34

Table 6.1 Number of convolutional layers versus accuracies of the network. ... 46

Table 6.2 Performance of the Mice-Mimic-Net under different sized kernels. ... 47

Table 6.3 Performance of the Mice-Mimic-Net under different number of convolutional filters. ... 47

Table 6.4 Performance of Mice-Mimic-Net under different number of fully connected layers. ... 48

Table 6.5 Performance of Mice-Mimic-Net under different dropout probabilities. 49

Table 6.6 Performance of Mice-Mimic-Net when trained with the pain label of a single facial element. ... 50

Table 6.7 Performance metrics of different SVMs. ... 57

Table 6.8 Performance of the K-NN on the mouse data set. ... 57

Table 6.9 Performance of the K-NN on the mouse data set when PCA is applied to the data set before classification is performed. ... 58
LIST OF FIGURES

FIGURES

Figure 3.1  An artificial neuron. ........................................ 9
Figure 3.2  A feed-forward neural network with one hidden layer. .... 10
Figure 3.3  The perceptron activation function. .......................... 11
Figure 3.4  The sigmoid activation function .............................. 12
Figure 3.5  The ReLU activation function. ............................... 12
Figure 3.6  A multi-layer neural network illustrating main notations which are
used to derive backpropagation algorithm. .............................. 18
Figure 3.7  Connection between local receptive field neurons and a first convol-
utional layer neuron. ..................................................... 22
Figure 3.8  Max pooling reduces the feature map size (here from 32x32 to 16x16). 23
Figure 3.9  A convolutional network with a single convolutional layer of 4
kernels resulting in 4 32x32 feature maps. Note that after 2x2 pooling,
feature map size is reduced to 16x16. ................................. 24
Figure 3.10 Polynomial and linear functions to fit noisy linear data. Linear
curve gives better generalization while polynomial curve has the problem
of over-fitting [1]. .................................................... 25
Figure 3.11 In drop-out, some nodes (neurons) of the network randomly and
temporarily deleted [2]. .............................................. 26
Figure 3.12 Separation of input into two classes by SVM. Samples on the margin are called the support vectors [3].

Figure 4.1 Different face side views of a mouse picture labeled as 1 to 6.

Figure 4.2 Faces without pain (pain scale of 0) and 6 different orientations.

Figure 4.3 Faces with moderate pain (pain scale of 1) and 6 different orientations.

Figure 4.4 Faces with severe pain (pain scale of 2) and 6 different orientations.

Figure 4.5 Pain degree observed in different organs of the mouse in the data-set.

Figure 5.1 General view of the Mice-Mimic-Net.

Figure 5.2 First convolutional layer outputs before and after max-pooling operation.

Figure 5.3 Fully connected and softmax layers of the Mice-Mimic-Net.

Figure 6.1 Test accuracy versus Adam optimizer learning rate in the network.

Figure 6.2 Data Augmentation: Training data size is doubled with artificially generating images by taking the horizontal symmetry of each images.

(a) original image

(b) vertical symmetry of the original image

Figure 6.3 Training and test accuracies. Train and test accuracies are 1 and 0.97 after 20k epochs, respectively.

Figure 6.4 Output error (cost) of the Mice-Mimic-Net goes down to 0.14.

Figure 6.5 Sample test image which is used to visualize convolutional layer outputs.

Figure 6.6 Filters and feature maps when the ultimate network is trained based on all the facial elements of the mouse.
Figure 6.7 Convolutional layers outputs thorough 1 to 6 when 2 more convolutional layers are inserted before max-pooling layers to the Mice-Mimic-Net. Note the low level to higher level details change from C1 through C6.  

Figure 6.8 Accuracies when multilayer perceptron network is used.  

Figure 6.9 Cross entropy cost when multilayer perceptron network is used.  

Figure 6.10 Accuracies when multilayer perceptron network is fed with PCA processed data.  

Figure 6.11 Performance comparison of the methods.  

Figure 6.12 ROC curve for class 0 (pain scale of 0 - painless) vs other two classes (moderate pain scale and severe pain scale) classification.  

Figure 6.13 ROC curve for class 1 (pain scale of 1 - moderate pain) vs other two classes (painless and severe pain scale) classification.  

Figure 6.14 ROC curve for class 2 (pain scale of 2 - severe pain) vs other two classes (painless and moderate pain scale) classification.  

Figure 6.15 Performances of the classification methods as the chunk size for constructing the train and test data changes. 20 percent of each chunk is chosen as test data.  

Figure 6.16 Performances comparisons of the classification methods when a new video is used as the test data.  

Figure 6.17 ROC curve for class 0 (pain scale of 0 - painless) vs other two classes (moderate pain scale and severe pain scale) classification when test data is composed of frames of a completely new video.  

Figure 6.18 ROC curve for class 1 (pain scale of 1 - moderate pain) vs other two classes (painless and severe pain scale) classification when test data is composed of frames of a completely new video.
Figure 6.19 ROC curve for class 2 (pain scale of 2 - severe pain) vs other two classes (painless and moderate pain scale) classification when test data is composed of frames of a completely new video.
CHAPTER 1

INTRODUCTION

1.1 Motivation

Medical experiments on laboratory mice have been helping humanity for many years. Pain recognition and assessment are some of these experiments. However, traditional methods of pain recognition and assessment that gives gross clinic signs are time consuming to conduct. Moreover, they are not specific to pain. Thus Langford et al [4] developed a manual pain assessment method called Mouse Grimace Scale (MGS), a standardized behavioral coding system with high accuracy and reliability. In Langford et al [4], laboratory mice were noxious stimulated and facial expression of pain is observed. This measurement of the pain provided insight into the subjective pain experience of laboratory mice. Although MGS technique is a time saving and robust in comparing to the other pain assessment techniques, it is still time consuming and relatively subjective.

1.2 Contribution

In this thesis, an automatic grimace scaling method for laboratory mice is developed using deep learning. Facial expressions of pain during noxious stimuli of moderate duration is evaluated by an artificial neural network. We believe such a automatic measurement technique provides more objective measurement results because evaluations made by more than one person are more likely to be objective. In this work,
experimental grimace scale evaluated by experts are trained to an convolutional neural network. Once the network is trained using this reliable inputs, utilizing the network for all of the later experiments is both time saving and more objective. Moreover, although there are some automated works on human facial expression classification, this work is the first computer based automated work performed for an animal (particularly mouse) facial expression (grimace) classification.

Deep learning refers to training procedure of artificial neural networks that are composed of several layers. It is a state-of-the-art machine learning technique and applicable to many domains of science and business [5]. Its application to many known machine learning tasks in speech and image recognition showed that it beats other state of the art techniques in this area [6, 7]. Although many methods which are used in deep learning today, had already been proposed several years ago, deep learning techniques became very popular with the advance in processing power of computers, availability of more data, and new architectures like convolutional neural networks [8]. Facial expression recognition, on the other hand, is a challenging machine learning task that has been attracting many research interests in this area. Many different techniques have been proposed to make machine learning algorithms close to human recognition system performance like Gaussian Face technique [9], FACS based techniques [10], support vector machines [11–15], Nearest Neighbor [16] and K-Nearest Neighbour algorithm [17]. However this approaches either needs pre-training or some local features of the image and are not close to human performance. On the other hand, deep convolutional networks have shown the state of the art performance in many image classification contest, such as ImageNet classification contest, and almost beats the human visual system [7]. In this thesis, a convolutional deep network called Mice-Mimic-Net is given a try on mouse face expression classification and results are highly satisfying; a 97 percent test accuracy is obtained. Performance of the Mice-Mimic-Net is also compared with some classical methods.

1.3 Organization of the Thesis

In the next chapter, related works performed on human facial expression action unit assignment, and computer based (automatic) classification techniques for this tasks
are investigated. Different classification algorithms used in these tasks are explored in the literature. The MGS (Mouse Grimace Scale) technique is also explained.

In chapter 3, basics of neural networks, the backpropagation technique used to train them, convolutional neural networks, and different training techniques and fine tuning methods are explained. Principal Component Analysis, Support Vector Machines (SVM), and K-Nearest Neighbour (K-NN) classification techniques are also shortly explained since they are used as base methods for performance evaluation of the Mice-Mimic-Net.

In chapter 4, the data set used in this work is introduced. It is constructed using MGS technique.

In chapter 5, architecture of the Mice-Mimic-Net is introduced.

In chapter 6, experimental results are given. In this chapter, the Mice-Mimic-Net is fine tuned using different parameters, architectures and techniques and result are shown. Moreover, performance of the Mice-Mimic-Net is compared those with K-Nearest Neighbour (KNN) method and Support Vector Machine (SVM) based classifiers.

Finally, conclusion and future work is given in chapter 7.
Facial expression coding in humans is well studied. There is a technique called Facial Action Coding System (FACS) [18] for human facial expression evaluation. This system taxonomizes human facial movements according to their appearance on the face. FACS is a common standard for labelling physical expression of emotions.

Although nonhuman mammals including mice shows facial emotional facial expressions, until now, there has been no research on facial expressions of pain in any nonhuman species. Thus Langford et al [4] developed the Mouse Grimace Scaling Method (MGS). This method consists of five facial action units (features). These features are orbital tightening, nose bulge, cheek bulge, ear position, and whisker change. Each of this regions shows a consistent changes as pain degree of the mouse changes, as explained in MGS Manual [19]. For example an eye squeeze is observed in the orbital muscles around the eyes (orbital tightening), and a bulge on top of the nose is observed (nose bulge) as the pain degree increases. Each action unit (AU) is categorized as 0 (AU not present), 1 (AU moderate) or 2 (AU severe). This units have been decided by facial pain expression experts as reliable indices of pain [4]. Overall pain degree of the mouse is calculated by averaging this 5 action units.

There are many experiments on human facial expression classification using computers. K-NN (K-Nearest Neighbor Classifier) is one of the methods used in this experiments [20–23]. In these works, human face is assigned actions units using the FACS method or similar methods, and facial emotion of this faces are tried to be classified like happy, angry, surprise, sad and neutral. Depending on whether the K-NN classifier is supported by some pre-processing technique and/or feature extraction,
classification accuracy obtained in this works scales between 80 to 90 percent [20–23].

On the other hand, Support Vector Machines (SVM) are also commonly used in human facial expression classification. In [12], a SVM classifier tries to predict the facial expressions of humans. Independent Component Analysis (ICA) is used as feature extraction method in this work. In [24], Michel et al used SVM to classify human face expression as coded in FACS technique and obtained an overall accuracy of 88 percent. The SVM classifier is also used in the works in [25–28] which perform automatic facial expression classification in videos. A Gabor Representation of the frame is formed initially, and then processed by a bank of SVM classifiers in majority of this works. Principal Component Analysis (PCA) is also a preferred pre-processing applied before SVM technique used in some works [29,30]. State of the art SVM works give up to 90 percent classification accuracy for human facial expression classification.

On the other hand, Khanam et al [31] used the fuzzy approach to classify five human facial expressions. [32–34] also show how to perform the facial expression recognition by rule-based methods.

Multi-Layer Perceptron (MLP) is also used for classifying human face expressions. In their 1996 paper [35] Padgett and Cottrell used a MLP to classify human face emotion labeled based on FACS technique and they obtained 86 percent of classification accuracy. This accuracy is bad comparing to today’s success of deep networks but it was a good result at that time when neural networks had poor performance compared to other classification techniques and were difficult to train. However with the advantage in the computers processing power and newly discovered training techniques and network architectures they become giving state of the art classification performance not only for facial expression classification but also in other classification tasks [6,7].

In their 2015 paper [36], Li et al used convolutional neural networks to classify CIFE (Candid Image Facial Expression) data set and they got 81 percent accuracy which outperform their SVM based baseline method having about 60 percent accuracy. In another 2015 work [37], Thai et al used Canny Edge Detector, PCA (Principal Component Analysis) and MLP (Multi-Layer Perceptron) to classify facial expression in JAFFE (Japanese Female Facial Expression) dataset and they obtained 86 percent accuracy which outperformed many previous works in this data set. In [38] Silva et
al obtained 99 percent classification accuracy in MUG (Multimedia Understanding Group) facial expression database [39] using combinations of K-NN, a multi-layer neural network (MLP) and Linear Discriminant Analysis (LDA). It is an automated task which consists of face detection, feature extraction and classification.

Since Langford et al [4], there was no any grimace scaling (pain assignment) technique for the animals. MGS is a relatively new method published in 2010, thus there are no computer based methods for it yet. In this thesis, a data set obtained from mice faces according to MGS manual is classified using CNN architecture and result are compared with baseline methods SVM and K-NN. This thesis is the first work performed on automatic grimace scaling not only for laboratory mice but also among other animals.
3.1 Artificial Neural Networks

3.1.1 Introduction

Artificial neural networks (ANNs) are a family of models inspired by biological neural networks in the central nervous systems of animals [40]. ANNs are more commonly used for regression and classification. This chapter gives a general review of the neural network concepts used in this thesis and highlights some of the most commonly used techniques and terms used today.

3.1.2 Artificial Neuron

An artificial neuron consist of inputs to itself, a bias and an output. Output function may be any function but usually sigmoid output or ReLU (Rectified Linear Unit) is preferred. Figure 3.1 shows an artificial neuron and its connections.

![Figure 3.1: An artificial neuron.](image-url)
Output of a neuron is calculated as a function of sum of its weighted inputs and bias. If we donate the weighted sum as \( z = \sum_j w_j x_j + b \) and neuron evaluation function as \( \sigma \), then \( a = \sigma(z) \) is output of this neuron.

3.1.3 Multi-Layer Neural Networks

Multi-Layer networks consist of many neurons which come together in a layered manner. There are two main types of artificial neural networks, \textit{feed forward nets} and \textit{recurrent nets}. Feed forward networks are many layer neural networks where neurons in a layer are only connected to next layer neurons in the network, in other words, they are the networks where connections between the neurons do not form a cycle. This kind of neural networks are also called as \textit{Multi-Layer Perceptron(MLP)}. A general view of the feed forward net is shown in 3.2.

![Multi-Layer Neural Network Diagram](image)

Figure 3.2: A feed-forward neural network with one hidden layer.
3.1.4 Neuron Output Functions

Recalling from section 3.1.2, an artificial neuron output is the function of its weighted input and bias. There are many forms of this evaluation functions and most commonly used forms are given in following sections.

3.1.4.1 Perceptron

It is probably the simplest activation function which is given as,

\[ \sigma(z) = \max(0, z). \]  

(3.1)

and shown in Figure 3.3

![Figure 3.3: The perceptron activation function](image)

Perceptron is used to classify binary inputs. A perceptron has a binary output. Learning procedure for a single perceptron is called perceptron convergence algorithm.

3.1.4.2 Sigmoid Function

Also known as log-sigmoid or logistic function and given by,

\[ \sigma(z) = \frac{1}{1 + e^{-z}}. \]  

(3.2)

Among the activation functions, this is the one close to representing the output of biological neurons. It has easy to compute derivative, that is, \( \sigma'(z) = \sigma(z)(1-\sigma(z)) \).
Saturating non-linearity as shown in the Figure 3.4 is one of the disadvantage of sigmoid function. It is clear that, as the training progress, learning of the network slows down.

![Figure 3.4: The sigmoid activation function](image)

### 3.1.4.3 ReLU

Its name comes from *rectified linear unit*. The function is given by,

\[
\sigma(z) = \max(0, z)
\]

which is plotted in Figure 3.5. Note that neurons with this activation function do not suffer from saturation.

![Figure 3.5: The ReLU activation function.](image)

### 3.1.5 Cost Functions

The main goal in the training of a neural network is to reduce some cost(error) function of the network for the given inputs. In this section, the most commonly
used error(cost) functions in order to calculate a neural network’s output error are explained. In the experiment, the soft-max model is used but others cost functions are also worth mentioning as they are widely used.

3.1.5.1 Quadratic Cost Function

It is one of the commonly used cost functions in neural networks output layer. It is also known as Mean Squared Error (MSE) and given by following equation

\[
C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.
\] (3.4)

where \(y(x)\) is the desired output of the network and \(a\) is the obtained output.

Since every term in the sum is non-negative, this function is always greater than zero. This function suffers from learning slowdown problem.

3.1.5.2 Cross Entropy Cost Function

It is an alternative to classical quadratic cost function which has learning slowdown problem. This happens when desired output and actual output values deviate from each others. In this case although learning rate should increase, it decreases. A good cost function should be able to handle this issue.

Cross-entropy cost function for a single neuron is given by

\[
C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)]
\] (3.5)

where \(x, y\) and \(a\) same as in equation 3.4. It can be shown that this expression is non-negative, that is \(C \geq 0\). Moreover, the cross-entropy will be close to zero when the neuron’s actual output is close to the desired output for all given inputs. For a single output network, the partial derivative of cross-entropy cost function with respect to weights and biases are given by equation 3.6 and 3.7, respectively.

\[
\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j(\sigma(z) - y).
\] (3.6)
\[ \frac{\partial C}{\partial b} = \frac{1}{n} \sum_x (\sigma(z) - y). \]  

(3.7)

The rate at which the weight is modified is controlled by \( \sigma(z) - y \), i.e., by the error at the output. Note that generalized version of cross-entropy function for multi-neuron output network can be given as

\[ C = -\frac{1}{n} \sum_x \sum_j [y_j \ln a^L_j + (1 - y_j) \ln(1 - a^L_j)] \]  

(3.8)

where \( L \) is the output layer in the network and \( a^l_j \) is the output of \( j \) the neuron in the \( l \) th layer. For more detailed explanation and full derivation of this cost function, see [2].

and derivatives of the cost function with respect to particular weight and bias is given in equation 3.9 and 3.10, respectively.

\[ \frac{\partial C}{\partial w^L_{jk}} = \frac{1}{n} \sum_x a^{L-1}_k (a^L_j - y_j) \sigma'(z^L_j). \]  

(3.9)

\[ \frac{\partial C}{\partial w^L_b} = \frac{1}{n} \sum_x (a^L_j - y_j) \sigma'(z^L_j). \]  

(3.10)

### 3.1.5.3 Soft-Max Regression

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to handle multiple classes. Logistic regression is a method to classify input vectors into two classes, namely 0 and 1. Soft-max regression is a more general form of the logistic regression, in which input vector is classified into more than one classes. In the experiment part of the thesis, soft-max output with 3 neurons is used. It tries to classify given input into 3 classes.

According to soft-max function, the activation \( a^L_j \) of the \( j \)th output neuron is given by

\[ a^L_j = \frac{e^{z^L_j}}{\sum_k e^{z^L_k}} \]  

(3.11)
It can be shown that, in soft-max output, summing all the soft-max outputs gives 1, that is
\[ \sum_j a^L_j = \frac{\sum_j e^{z^L_j}}{\sum_k e^{z^L_k}} = 1. \] (3.12)

### 3.2 Deep Learning

Deep learning is the training process of many layer feed forward neural networks. It is called deep, because there are many hidden layers in the network. Training this networks is relatively more difficult because of this deep layers. However some good algorithms exist to train them faster and most of them are built on the back-propagation Algorithm. In order to understand how this networks are trained, it is important to understand both gradient descent method and backpropagation algorithm which are given in the next chapters.

#### 3.2.1 Gradient Descent

Gradient descent is an optimization algorithm used in many fields. It is a first order optimization algorithm utilized to find local minimum of a differentiable function. It is also known as *steepest descent* algorithm. If we move towards the negative of the gradient of the function at the current point, we finally reach a local minimum. In neural networks, gradient descent method is applied on the cost function of the network. In a neural network training task, main goal is to minimize the cost function $C$ so that network output and desired output values can come close to each other. Note here that $C$ is a function of both $w$ and $b$, i.e; $C = C(w, b)$. If we let $W$ and $B$ weight and bias vectors of the network, respectively, then change in the cost function with respect to change in weight vector can be approximated as

\[ \Delta C \approx \Delta W \cdot \nabla C_W \] (3.13)
and similarly with respect to biases, it can be approximated as:

\[ \Delta C \approx \Delta B \cdot \nabla C_B \]  \hspace{1cm} (3.14)

if we choose \( \Delta W = \eta \nabla C_W \), where \( \eta \) is called learning rate, it is a small positive constant. Equation 3.14 can be re-written as:

\[ \Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \| \nabla C \|^2 \]  \hspace{1cm} (3.15)

Note here that because \( \| \nabla C \|^2 \geq 0 \), \( \Delta C \leq 0 \), i.e., \( C \) is a monotonically decreasing function of \( W \). Note also that similar equations holds for the bias vector \( B \).

Once above quantities are calculated, weights and biases are updated as , \( W \leftarrow W - \eta \Delta W \) and \( B \leftarrow B - \eta \Delta B \).

### 3.2.1.1 Stochastic Gradient Descent

In Gradient Descent Algorithm given in the previous chapter, in order to compute the gradient, we needed to first compute gradient separately for each input \( x \), then average them. This is a time consuming task, thus, in order to estimate average gradient, it is better to use not all but some of the individual gradient values. This gives an approximate result for the average gradient while saving big amount of computation time. If input group to be used for calculation of average gradient is chosen randomly among all training inputs, then this process is called stochastic gradient descent.

Putting it another way, we pick small number (\( m \)) of randomly chosen training samples among all of the samples then label them as \( X_1, X_2, \ldots, X_m \). This randomly chosen input group is named mini batch. When the sample size \( m \) is large enough it is expected that the average value of the \( \nabla C_{X_j} \) will be close to the averaging over all \( \nabla C_x \), that is,

\[ \frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n}, \]  \hspace{1cm} (3.16)

therefore

\[ \nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{X_j}, \]  \hspace{1cm} (3.17)
then weight and bias updates are computed as,

\[ w_k \leftarrow w_k - \frac{\eta}{m} \sum_j \frac{\partial C_X}{\partial w_k}, \]  

(3.18)

\[ b_l \leftarrow b_l - \frac{\eta}{m} \sum_j \frac{\partial C_X}{\partial b_l}, \]  

(3.19)

When all the training samples in the mini-batch are consumed in the above training process, a *training epoch* is completed and new training epoch is performed over and over again until the cost function gets small enough.

### 3.2.2 Backpropagation Algorithm

#### 3.2.2.1 Main Idea

In finding gradient, one utilizes the derivatives of the cost function with respect to biases and weights. Performing these operations at every step of weight and bias update from scratch is not cost effective. Fortunately, there is an algorithm called *backpropagation* which is nothing but utilization of the chain rule in calculus [41].

#### 3.2.2.2 Notation

- \( x \): The input vector.
- \( w_{jk} \): The weight from the \( k^{th} \) neuron in the \((l - 1)^{th}\) layer to \( j^{th} \) neuron in the \( l^{th} \) layer.
- \( b_k^l \): Bias of the \( k^{th} \) neuron in the \( l^{th} \) layer.
- \( a^l \) is the output vector for the \( l \) th layer of the network, \( b^l \) is the bias vector, \( w^l \) is the weight vector, \( z^l \) is the weighted input to the neurons in layer \( l \).
- \( y \): Desired output vector for given input vector \( x \).
- \( L \): Number of layers in the network.
\( \ominus \): The Hadamard product: \( s \odot t \) denotes the element-wise product of the two vectors. That means, \((s \odot t)_j = s_jt_j\).

\( \delta^l_j \): error term at the \( j^{th} \) neuron of \( l^{th} \) layer.

On the Figure 3.6, some of the weights and biases are shown for illustration. In this notation, \( w^k_j m \) is weight from \( j^{th} \) neuron from \( k^{th} \) layer to \( m^{th} \) neuron in the \((k-1)^{th}\) layer. Additionally, \( w^k_j \) shows the \( j^{th} \) neuron bias on the \( k^{th} \) layer. Note that neuron outputs are written inside circles(neurons).

Figure 3.6: A multi-layer neural network illustrating main notations which are used to derive backpropagation algorithm.

### 3.2.2.3 Derivation

Output vector of the \( l^{th} \) layer is

\[
a^l = \sigma(w^l a^{l-1} + b^l) = \sigma(a^l) \tag{3.20}
\]
Assuming the quadratic cost function given in section 3.1.5.1, it can be approximated as \( C = \frac{1}{n} \sum_x C_x \) over cost functions \( C_x \) by averaging training examples \( x \).

Defining the error term \( \delta_j^l \) of neuron \( j \) in layer \( l \) by

\[
\delta_j^l \equiv \frac{\partial C}{\partial z_j^l},
\]

(3.21)

Change of cost as a function of the \( j^{th} \) activation function is given as,

\[
\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).
\]

(3.22)

In matrix form it can be re-written as

\[
\delta^L = \nabla_a C \odot \sigma'(z^L).
\]

(3.23)

For the previously defined quadratic cost function it is equal to

\[
\delta^L = (a^L - y) \odot \sigma'(z^L).
\]

(3.24)

The error term at the previous equation can be written as

\[
\delta^l = ((w_l^{l+1})^T \delta_{l+1}^l) \odot \sigma'(z^l),
\]

(3.25)

The rate of change of the cost with respect to any bias in the network is

\[
\frac{\partial C}{\partial b_j^l} = \delta_j^l,
\]

(3.26)

Then change of the cost function with respect to the error term at the \( j^{th} \) neuron of \( L^{th} \) layer is

\[
\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)
\]

(3.27)

Finally, gradients for weights and biases are calculated as,
\[
\frac{\partial C}{\partial w_{jk}^l} = a_{k}^{l-1}\delta_{j}^{l}
\]
(3.28)

\[
\frac{\partial C}{\partial b_{j}^l} = \delta_{j}^{l}
\]
(3.29)

### 3.2.2.4 The Algorithm

What backpropagation provides us an easy computation of the gradient for the gradient descent algorithm. The backpropagation algorithm is given as:

1. **Apply an input from the training set.**

2. **Set the corresponding activation** \(a^1\) **for the input layer.**

3. **For each** \(l = 2, 3, \ldots, L\) **compute** \(z^l = w^l a^{l-1} + b^l\) **and** \(a^l = \sigma(z^l)\). **This is the feed-forward phase.**

4. **Calculate output error** \(\delta^L\), **and calculate the vector** \(\delta^L = \nabla_a C \odot \sigma'(z^L)\).

5. **For each** \(l = L - 1, L - 2, \ldots, 2\) **compute** \(\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)\). **This is the backpropagation phase.**

6. **Calculate the gradient of the cost function with respect to weights and biases as**

\[
\frac{\partial C}{\partial w_{jk}^l} = a_{k}^{l-1}\delta_{j}^{l} \quad \text{and} \quad \frac{\partial C}{\partial b_{j}^l} = \delta_{j}^{l}.
\]

7. **For each** \(l = L, L-1, \ldots, 2\) **update biases and weights according to the equations** 3.30 and 3.31, **respectively. This step corresponds to gradient descent phase.**

\[
w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T
\]
(3.30)

\[
b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}
\]
(3.31)
This step corresponds to gradient descent.

8. Repeat above steps until the output error (cost function) gets small enough.

3.2.3 Convolutional Neural Networks

3.2.3.1 Motivation

Traditional multilayer perceptron model suffers from the curse of dimensionality due to the full connectivity between neurons. Moreover, this traditional architecture does not take the spatial structure of the input data into account, that is, it treats input pixels which are far apart or close together in a same manner. Apparently, the full connectivity of neurons is wasteful, and the huge number of parameters quickly lead to problem of over-fitting.

Convolutional Neural Networks (CNN) are inspired from local receptive field of monkey’s visual cortex [42]. When monkeys are shown familiar faces, monkey brain lights up in a specific area as observed in the functional magnetic resonance imaging (fMRI) system. Receptive cells in the monkey visual system are sensitive to small sub-regions of the visual field, called a receptive field, whereas in the convolutional nets, local receptive fields on the image are connected to the individual neurons in the first hidden layer. CNNs use spatially local correlation between neurons of adjacent layers by enforcing a local connectivity pattern, thus learnt filters produce the strongest response to a spatially local input pattern.

3.2.3.2 Architecture

In fully-connected feed-forward nets, every pixel of the image is connected to the every neuron of the first hidden layer of the network. On the other hand, in CNN architecture, only local regions of the image are connected to the each hidden layer neurons. That region in the input image is called the local receptive field for the hidden neuron. This local receptive field is slided through full image, which corresponds to convolving receptive field and input. Slide operation is usually done one pixel at a time, but other stride lengths are also possible. Weights and biases are shared among the
receptive fields, that is each neuron in the first hidden layer has same weights and bias but connected to the different local area pixels. Shared weights and bias are called kernel or filter. This means that each neuron in the first layer tries to detect the same feature but at different locations. That means convolutional networks are well suited to finding patterns with location variance. Another advantage of sharing weights and biases is that it significantly reduces the number of parameters involved in a network.

CNNs inputs are not restricted to 2 dimension. One can use 3 dimensional convolutional layers for MRI data or 1 dimensional layers for audio data. In this cases, filters would become appropriately dimensioned, and pooling layers would change dimension as well.

Figure 3.7 illustrates a local 3x3 receptive field connected to the first hidden layer (convolutional layer) neuron.

Figure 3.7: Connection between local receptive field neurons and a first convolutional layer neuron.

Assuming local receptive field of size $m \times m$ where $m + 2k + 1$, activation value of a hidden layer neuron is

$$f(b + \sum_{l=-k}^{k} \sum_{m=-k}^{k} w_{l,m} a_{j+l,k+m})$$

(3.32)

where $f$ is the non-linearity applied at the neuron output and may be chosen as sigmoid or ReLU function.

Since there are usually more than one feature of the image to be extracted, there are more then one filter and corresponding feature map in the first layer, usually many. In addition to the convolutional layers just described, convolutional neural networks
also contain pooling layers. They are used after convolutional layer and simplify the information in the output from the convolutional layer. Max pooling is a common pooling technique. An 2x2 max pooling layer, for example, selects output value having the biggest value among the 4 adjacent neuron outputs. If we apply max pooling to \( m \) by \( n \) feature map output, for instance, its size is reduced to \( m/2 \) by \( n/2 \). Figure 3.8 illustrates max pooling operation by 2x2, after first convolutional layer.

![Figure 3.8: Max pooling reduces the feature map size (here from 32x32 to 16x16).](image)

Another commonly used pooling technique is \( L2 \) pooling. It is the square root of the sum of the squares of the activations in the 2 by 2 region.

After one or more convolutional plus max pooling layers, there comes the fully-connected layer. Every neuron from the max-pooled layer is connected to the each neuron of the fully connected layer by the fully connected layer. There might be more than one fully connected layers in MLP.

Figure 3.9 shows the final appearance of a convolutional network with input neurons of 36x36, 4 feature map of 32x32, pooling size 2x2 and output layer neurons.

Convolutional nets usually have more than one convolutional layer and each convolutional layer has many kernels. For example, in their network, Lenet-5, Hinton et al [7] used five convolutional and three fully-connected layers with each convolutional layer having about 100 filters.
Figure 3.9: A convolutional network with a single convolutional layer of 4 kernels resulting in 4 32x32 feature maps. Note that after 2x2 pooling, feature map size is reduced to 16x16.

3.2.3.3 Backpropagation in Convolutional Layers

Backpropagation algorithm is almost same for convolutional networks. The only difference is that, since convolutional layers have shared weights, all expressions contributing to a weight are summed before taking derivative. Thus, we need to replace individual neuron’s activation value by equation 3.32, and update other equations accordingly.

3.2.3.4 Backpropagation in Max-Pooling Layers

The max-pooling layers do not actually do any contribution to learning. In forward propagation, a neuron in this layer chooses its output as the maximum of its inputs. This output value acquires an error computed from backward propagation from the next layer. This error is then just conveyed backward to the previous layer neuron whose output is accepted in the forward step.

3.2.4 Techniques to Improve Training Performance

A common problem that occur during neural network training is the over-fitting problem. After enough number of training epochs, error on the training data sets gets small enough, but when the test data is fed to the network the error is larger. This shows that the network has memorized the training examples, but it has not learned to
generalize new situations, i.e., to test data. One method for preventing over-fitting is to use a network which has optimum size to provide an adequate fit. As the network gets larger, it can represent more functions and it over-fits the training data. On the hand, if it is too small, it cannot learn the function. Thus, the size of the network should be enough to learn, but should not be too large to over-fit.

Another way to avoid over-fitting is to use large amount of data. If we use large data set to train the network, then it will not have enough power to memorize inputs. Figure 3.10 illustrates an over-fitting example. Roughly linear and noisy data is fitted to both linear and polynomial functions. Although the polynomial function is an almost perfectly fitted to input data, the linear version can be expected to generalize better.

The data set used in this work is large enough and network is shallow to generalize. Nonetheless, to make it generalize better, there is another technique called Dropout which is given in the following section.

Another related problem faced when training deep networks is the time it takes. Although backpropagation technique boosts traditional gradient descent method, there is a more evolved version called Adam Optimizer. This technique is given in section 3.4.2.

Figure 3.10: Polynomial and linear functions to fit noisy linear data. Linear curve gives better generalization while polynomial curve has the problem of over-fitting [1].
3.2.4.1 Dropout

In drop out technique network itself is modified, not the cost function. Pre-chosen amount of the hidden neurons in the network are randomly deleted, while leaving the input and output neurons same. Then input is forwarded through modified network while backpropagation is performed through original network and weight update is performed on non-deleted connections. When training is done by dropout, overall weights should be halved. Dropout technique is especially useful in training deep networks, where the problem of over-fitting is experienced. The neurons which are “dropped out” in this way do not contribute to the forward pass and do not participate in back-propagation.

Figure 3.11 illustrates dropout in a network.

![Figure 3.11: In drop-out, some nodes (neurons) of the network randomly and temporarily deleted [2].](image)

3.2.4.2 Using Adam Optimizer

There are many variants of the stochastic gradient descent method, all having the same main idea as gradient descent, but performs parameter updates not only on the learning rate and gradient of the cost, but also on some additional parameters such as momentum and moving average of the gradient [43]. Adam Optimizer computes a decayed moving average of the gradient and the squared gradient at each time step.
Its first order moment coefficient is decayed over the time. Since first and second order moments are initially set to zero, some bias-correction is used to counteract the resulting bias towards zero. The use of the first and second order moments, in most cases, ensure that the gradient descent step size is typically $\approx \pm \eta$ and that in magnitude it is less than $\eta$. Let $\theta_t$ be a generic parameter of the network which is to be optimized (weight or bias). The goal here is to optimize this parameter according to error (lost) function of the network. Given hyper-parameters $\gamma_1$, $\gamma_2$, $\lambda$, and $\eta$, and setting $m_0 = 0$ and $g_0 = 0$, the update rule is as follows [43]:

$$m_{t+1} = \gamma_1 m_t + (1 - \gamma_1) \nabla L(\theta_t)$$

$$g_{t+1} = \gamma_2 g_t + (1 - \gamma_2) \nabla^2 L(\theta_t)^2$$

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \gamma_1^{t+1}}$$

$$\hat{g}_{t+1} = \frac{g_{t+1}}{1 - \gamma_2^{t+1}}$$

$$\theta_{t+1} = \theta_t - \frac{\eta \hat{m}_{t+1}}{\sqrt{\hat{g}_{t+1}} + \epsilon}$$

### 3.3 Kernel Methods

Kernel methods are the algorithms that are used for classification, regression or pattern analysis tasks. They do not need feature vector representations like non-kernel methods. In other words, this methods do not need user specified feature maps, they only need user specified kernels. A kernel can be described as a similarity function between inputs vector of the data to be classified [44]. In a typical machine learning pipeline, for example for an image classification task as in this work, without utilizing kernel methods, first step would be computation of the feature vector of each image, second step could be feeding this feature vectors and labels into a learning algorithm. On the other hand, in a kernel method, a single kernel function is employed in order to compute similarity between input vectors. Thus, for a classification task with kernel methods, like the one in this work, requirements are input images, labels, and a suitable kernel. Kernel methods are easy to use and have given state of the art results since they first appeared [45]. Although kernel methods are not in main interest of this thesis,
in chapter 5, there is a need for comparing the performance of the Mice-Mimic-Net with some state of the art methods. Because of both their success and simplicity, kernel methods are chosen for the comparison. Although there exist many kernels methods [46], Mice-Mimic-Net is compared with most commonly used techniques, namely; SVM, PCA pre-processed SVM and PCA pre-processed K-NN (K-NN is not a kernel method but PCA is). These methods are briefly explained in this chapter. Further details can be found in given references in each section.

3.3.1 Support Vector Machines

Support Vector Machine (SVM) is a classifier method that performs classification tasks by constructing hyperplanes in a multidimensional space of multidimensional class labels. Given labeled training data, it tries to construct an optimal hyperplane which categorizes new examples. SVMs normally can classify input data into two groups. For multiple class classification more than one SVMs are used. SVM uses an iterative training algorithm, which is used to minimize an error function to construct an optimal hyperplane. The training examples that are closest to the hyperplane are called support vectors. They are the elements of the training set that would change the position of the dividing hyperplane if removed. SVM can be used both in classification and regression tasks. SVM can be thought as a non-probabilistic binary linear classifier. SVMs can also perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. This transform SVMs to a kernel method indeed it is actually a linear classifier.

3.3.2 Formulation

This formulation steps are compiled from [47] and more details can be found there.

An hyperplane can be represented as

\[ y = w^T x + b \]  \hspace{1cm} (3.33)

where \( w \) is the weight vector and \( b \) is the bias. Hyperplane given in equation 3.33 can be represented in an infinite number of different ways by scaling \( w \) and \( b \). As a
convention, canonical hyperplane representation

\[ |w^T x + b| = 1 \]  \hspace{1cm} (3.34)

is usually chosen. Here, \( x \) represents support vectors. For the canonical hyperplane, the distance to the support vectors is given by equation 3.35.

\[ \frac{|w^T x + b|}{||w||} = \frac{1}{||w||} \]  \hspace{1cm} (3.35)

Figure 3.12: Separation of input into two classes by SVM. Samples on the margin are called the support vectors [3].

If we donate the margin between closest input vectors, \( M \), then

\[ M = \frac{2}{||w||} \]  \hspace{1cm} (3.36)

The goal here is to maximize the \( M \), in others words, maximize the distance between two closest support vectors of the different sides of the hyperplane which is equivalent to minimizing the function \( L(w) \) as given in equation 3.37.

\[ \min_{w,b} L(w) = \frac{1}{2} ||w||^2 \text{ subject to } y_i(w^T x_i + b) \geq 1 \forall i \]  \hspace{1cm} (3.37)

3.3.3 Principal Component Analysis

*Principal Component Analysis* is a method to discover or to reduce the dimensionality of the data set. In other words, it is a transform of data set to a new one, usually
having less dimensions, to better explore meaningful underlying variables. Among
the transformed components, first principal component gives as much of the variability
in the data as possible, and each succeeding component gives as much of the remaining
variability as possible.

### 3.3.3.1 Formulation

Assuming input vector $X$ and output vector $Y$ with each having size $p$, each output $Y_i$
can be expressed as linear combinations of inputs, that is to say,

$$Y_i = e_{i1}X_1 + e_{i2}X_2 + \cdots + e_{ip}X_p$$  \hspace{1cm} (3.38)

Here $Y_i$s are the **principal components**. The first principal component is the linear
combination of $X$ vectors that has maximum variance among all the linear combina-
tions. It contains much variation in the data as possible. Second principal component
is linear combinations of $X_i$’s that contains as much remaining variance as possible,
with a constrain that it is uncorrelated with the first principal component. The $i^{th}$
component will contain as much variations as possible remaining variance, which is
less than preceding $i - 1$ principal component variances and it will be uncorrelated
with all of them. There is one more constraint that each principal component has
the sums of squared coefficients which adds up to one. Expressing mathematically,
following equations should be satisfied.

$$\text{var}(Y_i) = \sum_{k=1}^{p} \sum_{l=1}^{p} e_{ik}e_{il}\sigma_{kl} = e_i^T\Sigma e_i$$  \hspace{1cm} (3.39)

where $\sigma_{kl}$ is the population variance-covariance between $k^{th}$ and $l^{th}$ inputs.

$$e_i^Te_i = \sum_{j=1}^{p} e_{ij}^2 = 1$$  \hspace{1cm} (3.40)
\[
\text{cov}(Y_{i-1}, Y_i) = \sum_{k=1}^{p} \sum_{l=1}^{p} e_{i-1,k} e_{il} \sigma_{kl} = e_{i-1}^T \Sigma e_i = 0 \quad (3.41)
\]

The steps in principal component analysis are given below:

1. **Compute the covariance matrix of the whole input data set.**

2. **Compute eigenvectors and corresponding eigenvalues for each input.**

3. **Sort the eigenvectors by decreasing eigenvalues and choose the specified amount of eigenvectors among the eigenvectors computed in step 2.**

4. **Use chosen eigenvectors to transform the inputs to the new input space.**

### 3.4 K-Nearest Neighbors Algorithm

The k-means clustering algorithm separates given input vectors into a fixed number (k) of clusters. It classifies a new coming input using similarity measure of the input and all other inputs. This method does not attempt to construct a general internal model, but it stores instances of the training data. Classification is made using majority vote (based on distance) of the nearest neighbors of currently classified points. The distance between an input vector and other stored (already having labels) inputs can be measured using different distance functions such as Euclidean distance or Manhattan distance but classical Euclidean distance is more common. Assuming \( n \) input vector \( \mathbf{x}_i \) \( p \) with feature vector \( (x_{i1}, x_{i2}, \ldots, x_{ip}) \), the Euclidean distance between sample \( \mathbf{x}_i \) and \( \mathbf{x}_l \) \( (l = 1, 2, \ldots, n) \) is computed as

\[
d(\mathbf{x}_i, \mathbf{x}_l) = \sqrt{(x_{i1} - x_{l1})^2 + (x_{i2} - x_{l2})^2 + \cdots + (x_{ip} - x_{lp})^2}. \quad (3.42)
\]

The steps at the K-NN algorithm is given below:
1. Go through each item in my data-set, and calculate the distance from that data item to new input.

2. Classify the sample as the majority class between $K$ samples in the data-set having minimum distance to the sample.
CHAPTER 4

THE MOUSE GRIMACE DATA-SET

This is the data set containing train data which is used to train the mouse-net (given in chapter 5) and validating test set. It contains images of the mouse faces in grayscale. Images are captured from HD videos and labeled as explained in the Mouse Grimace Scale (MGS) manual [19] in 256x256 format. Images are labeled by medical experts into 3 classes with respect to pain degree of the mice: none, moderate and severe. Medical experiment on the mice (drug stimuli, etc.) was performed according to the technique given in [19] by medical experts. After labeling, each frame is embedded with corresponding label attached to it into the numpy compressed file format (npz) [48]. There are 10k items in the data-set. An item consists of an image and corresponding pain label. Note that images are down-sampled to 64 by 64 and gray scaled before embedding into the database in order to shorten training time of the network.

There are six different views of the mouse faces in the databases and for each view there are 3 pain scales. Different face views of the mice are illustrated in figure 4.1. Face orientation information of the frames is not used in this thesis, but for the sake of complicity and for a possible future usage by other works, it is given. Some mouse face samples from the data set with different orientations (Face Sides - FS) and pain degree are given in the figures 4.2 to 4.4.

The mouse data-set contains three types of data objects in the compressed file; an image array, corresponding labels for this images, and the size of data-base. Each Label object contains information about where the face is looking at and the pain
degree of the face for 5 kinds of organs: eyes, ears, nose, mouth and mustache. Each property has values between 0 to 2 with ascending pain degrees. The table 4.1 shows corresponding property for each label object index.

Table 4.1: Each item of the label array in the data-set holds distinct features about corresponding face in the database.

<table>
<thead>
<tr>
<th>index</th>
<th>property</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>face sight</td>
<td>$FS$</td>
</tr>
<tr>
<td>2</td>
<td>eyes pain scale</td>
<td>$PS_{eye}$</td>
</tr>
<tr>
<td>3</td>
<td>ears pain scale</td>
<td>$PS_{ears}$</td>
</tr>
<tr>
<td>4</td>
<td>nose pain scale</td>
<td>$PS_{nose}$</td>
</tr>
<tr>
<td>5</td>
<td>mouth pain scale</td>
<td>$PS_{nose}$</td>
</tr>
<tr>
<td>6</td>
<td>mustache pain scale</td>
<td>$PS_{mustache}$</td>
</tr>
</tbody>
</table>
Figure 4.2: Faces without pain (pain scale of 0) and 6 different orientations.

Figure 4.3: Faces with moderate pain (pain scale of 1) and 6 different orientations.
Figure 4.4: Faces with severe pain (pain scale of 2) and 6 different orientations.

Figure 4.5 shows the pain labels of individual items in the data set extracted from a video during a pain experiment. As it can be seen from the figure, individual pain labels of face elements are strongly correlated with each others. After around 2500\textsuperscript{th} label, pain of the mouse is switched from basal (no pain) to moderate pain and progressively to the massive pain.

4.1 Usage

In python environment, opening a npz compressed file and extracting items from it trivial as given in Listing 4.1.

Listing 4.1: Code sample to load mouse data set in a python program.

```python
import numpy as np

npzfile = np.load(path/to/dataset/mousetrain.npz)
npztestfile = np.load(path/to/dataset/mousetest.npz)
TrainingImageCount, TrainingImages, TrainingLabels = npzfile[size], npzfile[Im], npzfile[lbl]
```
Figure 4.5: Pain degree observed in different organs of the mouse in the data-set.

TestImageCount, TestImages, TestLabels = npztestfile[size], npztestfile[Im], npztestfile[lbl]
CHAPTER 5

THE MICE-MIMIC-NET

5.1 Network Architecture

Mice-Mimic-Net is a relatively small convolutional neural net. It has 2 convolutional layers, 2 max-pooling layers one for each convolutional layer, a fully connected layer after convolutional and max pooling layers, and an output layer with 3 soft-max outputs. The Mice-Mimic-Net is given in Figure 5.1. Details of each layer is explained in the following section.

Inputs to the network are the images from the mouse data-set given in chapter 4. Each image has size of 64x64 and transformed to 1 dimension. That means, there are 64x64 input neurons in the Mice-Mimic-Net. Following sections explains connections between its layer and task of each layer.

5.1.1 Input Layer

It consists of 64x64 input neurons. Each neuron is connected to a pixel of the image, and to a bias of normal distribution with a standard deviation of 1.0. This layer neurons are connected to the first convolutional layer via kernels of size 10x10. There are 32 convolutional kernels(filters) between input layer and the first convolutional layer.
5.1.2 First Convolutional and Max Pooling Layer

This layer consist of 32 convolutional kernels of each $10 \times 10$ and produces $32 \times 64 \times 64$ (32 outputs each at $64 \times 64$) feature maps (images). The result of $64 \times 64$ image convolution with $10 \times 10$ kernel is cropped so that output is kept in the size of input, that is $64 \times 64$. Neurons have sigmoid output function.

After the convolutional output, a max-pooling filter(layer) is applied. This filter takes the one with highest value of 4 neighbour pixels and produces $32 \times 32$ feature maps, reducing convolutional layer output from $64 \times 64$ to $32 \times 32$ as shown in Figure 5.2.
5.1.3 Second Convolutional and Max Pooling Layer

This layer consists of 32 convolutional kernels each having size 5x5x32. It takes the output of first convolutional-max-pooling layer and produces 32@32x32 feature maps. Similar to the first convolutional layer, there follows a max-pooling layer of size 2x2, reducing the output from convolutional layer to 32@16x16. Outputs of this layer are fed into the fully connected layer.

5.1.4 Fully-Connected Layer

Finally, after 2 convolutional and max pooling layers, the high-level reasoning in the neural network is done via this layer. There are 32 kernels in the second convolutional layer, thus there are 32x16x16 neuron outputs after the second max-pooling layer. Each of them is connected to the each neuron of this fully connected layer of size 1000. Like all layers, sigmoid is chosen as activation function for each neuron. Dropout is applied just after this layer to overcome over-fitting issue. Note that, connecting any convolutional layer after this layer is meaningless because we lost spatial features once connected images (convolutional layer outputs are also images) to this layer. Figure 5.3 shows connection between fully connected layer and soft-max layer of the mouse-net.

![Diagram](image)

Figure 5.2: First convolutional layer outputs before and after max-pooling operation.
5.1.5 The Soft-Max Output Layer

This is the last layer of the mouse-net. It has 3 neurons and each neuron is fed from all the 1000 neurons of the previous layer (fully connected layer). As mentioned previously, the sum of the soft-max neuron outputs results in 1. During the training of the Mice-Mimic-Net, one hot shot form output is used. That is, neuron output with highest soft-max value is taken as 1, and other two as 0.

5.2 Why This Network

There are some reasons why Mice-Net utilizes convolutional neural networks instead of MLP. First of all, fully connected layers do not take spatial features of the input
when it is an image. Each pixel of the image is treated in same way, ignoring
the spatial correlation between images. On the other hand, convolutional networks
make the explicit assumption that the inputs are images, which allows encoding certain
properties into the architecture. Input images of the mouse face are space independent,
that is, although it is cropped as 256x256 from the video, it is not necessarily centered
as in the mnist [49] data set. Moreover, it makes the forward function more efficient to
implement and reduces the amount of parameters in the network. Hence, convolutional
layers are preferable for this work.

Since fully connected layer output does not contain spatial features anymore, i.e, it is
one dimensional, there is no use to put any convolutional layer after the fully connected
layer. On the other hand, as we will see in the experiment chapter, putting more fully
connected layer after convolutional layers neither increases the training accuracy nor
the test accuracy, but increases the computational load, hence the training time. Thus,
one fully connected layer is preferred in the Mice-Mimic-Net.
CHAPTER 6

EXPERIMENTAL WORK

In this section, effects of changing the network layers and parameters are presented. Unless stated otherwise, experiments are performed using the random 10 percent of the mouse data as test data, and the rest as the training data and each measurement was made across 20k training epochs with epoch size of 200. Whenever some parameters would be bench-marked, others parameters are kept same as in the Mice-Mimic-Net given in chapter 5. Experiments and tricks which resulted in the Mice-Mimic-Net are explained.

6.1 Choosing Network Parameters

6.1.1 Adam Optimizer Learning Rate

Mice-Mimic-Net’s weights and biases are updated using Adam Optimizer Algorithm, a different flavour of the backpropagation algorithm. Adam optimizer is explained in chapter 3.2.4.2. Figure 6.1 shows Adam learning coefficient versus test accuracy. It is seen that optimum learning rate for this optimizer is around 0.0001. Test accuracy decreases sharply when learning rate is around 0.0015 and stays below 0.5 after that point.

6.1.2 Number of Convolutional Layers

After two convolutional layers, although training time increases significantly, training and test accuracies almost stay the same. Thus there is no use to add more
Figure 6.1: Test accuracy versus Adam optimizer learning rate in the network.

convolutional layers to Mice-Mimic-Net. Two convolutional layer solution is almost optimal. Table 6.1 shows the effect of number of convolutional layers on the training and test accuracies. Note that after each convolutional layer there is also has a 2x2 max pooling layer. Table 6.1 reveals that number of convolutional layers do not have important effect on accuracy of the network. Note that kernel size is kept 5x5 at each convolutional layer.

Table 6.1: Number of convolutional layers versus accuracies of the network.

<table>
<thead>
<tr>
<th>number of convolutional layers</th>
<th>train accuracy</th>
<th>test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.997</td>
<td>0.950</td>
</tr>
<tr>
<td>2</td>
<td>0.999</td>
<td>0.979</td>
</tr>
<tr>
<td>3</td>
<td>0.955</td>
<td>0.945</td>
</tr>
<tr>
<td>4</td>
<td>0.953</td>
<td>0.940</td>
</tr>
</tbody>
</table>

6.1.3 Kernel Size

Table 6.2 shows network performance with various kernel sizes. Using 10x10 kernel for the first convolutional layer and 5x5 kernel for the second convolutional layer gives the best test accuracy.
Table 6.2: Performance of the Mice-Mimic-Net under different sized kernels.

<table>
<thead>
<tr>
<th>layer1 kernel size</th>
<th>layer2 kernel size</th>
<th>train accuracy</th>
<th>test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>2x2</td>
<td>0.976</td>
<td>0.957</td>
</tr>
<tr>
<td>5x5</td>
<td>5x5</td>
<td>0.993</td>
<td>0.950</td>
</tr>
<tr>
<td>10x10</td>
<td>5x5</td>
<td>0.989</td>
<td>0.97</td>
</tr>
<tr>
<td>10x10</td>
<td>10x10</td>
<td>1.0</td>
<td>0.963</td>
</tr>
<tr>
<td>15x15</td>
<td>10x10</td>
<td>0.99</td>
<td>0.933</td>
</tr>
<tr>
<td>15x15</td>
<td>15x15</td>
<td>0.999</td>
<td>0.966</td>
</tr>
</tbody>
</table>

6.1.4 Number of Filters

Table 6.3 shows the effects of changing number of filters (kernels) at each convolutional layers. 32 filters for each convolutional layers seems good fit. Trainings are done with 10k epochs.

Table 6.3: Performance of the Mice-Mimic-Net under different number of convolutional filters.

<table>
<thead>
<tr>
<th>filters amount(1st/2nd layers)</th>
<th>training time(sec)</th>
<th>train accuracy</th>
<th>test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>64/64</td>
<td>1532</td>
<td>0.979</td>
<td>0.947</td>
</tr>
<tr>
<td>64/32</td>
<td>1225</td>
<td>0.973</td>
<td>0.958</td>
</tr>
<tr>
<td>32/64</td>
<td>905</td>
<td>0.996</td>
<td>0.955</td>
</tr>
<tr>
<td>32/32</td>
<td>713</td>
<td>0.998</td>
<td>0.969</td>
</tr>
<tr>
<td>32/16</td>
<td>749</td>
<td>0.995</td>
<td>0.961</td>
</tr>
<tr>
<td>16/32</td>
<td>486</td>
<td>1</td>
<td>0.90</td>
</tr>
<tr>
<td>16/16</td>
<td>415</td>
<td>0.975</td>
<td>0.953</td>
</tr>
<tr>
<td>16/8</td>
<td>395</td>
<td>0.96</td>
<td>0.945</td>
</tr>
<tr>
<td>8/16</td>
<td>359</td>
<td>0.985</td>
<td>0.947</td>
</tr>
<tr>
<td>8/8</td>
<td>338</td>
<td>0.958</td>
<td>0.954</td>
</tr>
</tbody>
</table>
6.1.5 Number of Fully Connected Layers

Increasing number of fully connected layers increases training time but not accuracy. Table 6.4 shows the effect of number of fully connected layers on training time and accuracy. First fully connected layer has size 1024 and at each preceding layer, size is halved. As seen in table 6.4 one fully connected layer is enough for the Mice-Mimic-Net. Note that adding more layers increases the network parameters, which makes the optimization harder, thus training accuracy is increases. Since fitting for all parameters cannot be achieved, over-fitting is also decreases, resulting in increased test accuracy.

Table 6.4: Performance of Mice-Mimic-Net under different number of fully connected layers.

<table>
<thead>
<tr>
<th>fully connected layers</th>
<th>training time(sec)</th>
<th>train accuracy</th>
<th>test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1116</td>
<td>1</td>
<td>0.958</td>
</tr>
<tr>
<td>2</td>
<td>1088</td>
<td>1</td>
<td>0.953</td>
</tr>
<tr>
<td>3</td>
<td>1088</td>
<td>0.998</td>
<td>0.958</td>
</tr>
</tbody>
</table>

6.1.6 Dropout Probability

Although this method increases the training time, it overcomes the problem of over-fitting (memorization) as explained in Chapter 3, thus it increases the test accuracy which is the main goal of the Mouse-Net. Dropout is applied between fully connected and soft-max layers. In principle, dropout could be applied to the convolutional layers. However, convolutional layers have considerable inbuilt resistance to over-fitting [50]. This is because of shared weights. Shared weights force convolutional filters to learn from across the entire image. This makes them less likely to pick up on local features in the training data. Thus there is no need to apply dropout after convolutional layers. Table 6.5 gives performance of Mice-Mimic-Net with different dropout probabilities.
Table 6.5: Performance of Mice-Mimic-Net under different dropout probabilities.

<table>
<thead>
<tr>
<th>dropout probability</th>
<th>train accuracy</th>
<th>test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>0.97</td>
<td>0.947</td>
</tr>
<tr>
<td>0.2</td>
<td>0.988</td>
<td>0.957</td>
</tr>
<tr>
<td>0.3</td>
<td>1.0</td>
<td>0.962</td>
</tr>
<tr>
<td>0.4</td>
<td>1.0</td>
<td>0.958</td>
</tr>
<tr>
<td>0.5</td>
<td>0.998</td>
<td>0.971</td>
</tr>
<tr>
<td>0.6</td>
<td>1.0</td>
<td>0.959</td>
</tr>
<tr>
<td>0.7</td>
<td>1.0</td>
<td>0.951</td>
</tr>
<tr>
<td>0.8</td>
<td>1.0</td>
<td>0.959</td>
</tr>
<tr>
<td>0.9</td>
<td>0.943</td>
<td>0.937</td>
</tr>
</tbody>
</table>

6.2 Training With Single Facial Elements

In the final work, and in the previous experiments, Mice-Net is trained using pain information obtained from all five facial elements of the mouse. Average pain degree of these facial elements were fed into the Mouse-Net with corresponding image. Although this approach works well, it is worth to see how Mice-Mimic-Net would behave if it is trained using pain label of single facial element.

For training the network with each individual facial element; cost value (that is the error at network output), training and test accuracies are shown in table 6.6. It is apparent that, network learns much faster when trained based on a single thing rather then pain degree of all facial elements. This results in a quite fast learning network. The reason is simple; it tries to learn less things, particularly a single thing. Note that training and test accuracies are around 99 and 95 percent even after 1000 epochs. Table 6.6 also shows that, although individual facial elements tell us pain degree less accurate, merging the information in all of them gives us better accuracies. This is probably, the networks is compensating wrong information of an facial element with others when taking the average of all.
Table 6.6: Performance of Mice-Mimic-Net when trained with the pain label of a single facial element.

<table>
<thead>
<tr>
<th>training based on</th>
<th>train accuracy</th>
<th>test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>eyes</td>
<td>0.985</td>
<td>0.939</td>
</tr>
<tr>
<td>ears</td>
<td>0.979</td>
<td>0.946</td>
</tr>
<tr>
<td>nose</td>
<td>0.978</td>
<td>0.946</td>
</tr>
<tr>
<td>mouth</td>
<td>1</td>
<td>0.958</td>
</tr>
<tr>
<td>moustache</td>
<td>0.943</td>
<td>0.937</td>
</tr>
</tbody>
</table>

6.2.1 Data Augmentation

Data augmentation stands for artificially producing additional training samples in machine learning terminology. This can be done in many different ways. Here, it is simply employed the horizontal mirror symmetry of the training data to augment it as shown in Figure 6.2. The test accuracy of the network trained for the same number of epochs (20k), is increased about 1 percent. This is not a value to underestimate especially when the network accuracy is already above 90 percent.

![Figure 6.2: Data Augmentation](image)

Figure 6.2: Data Augmentation: Training data size is doubled with artificially generating images by taking the horizontal symmetry of each images.
6.3 The Final Network

Once getting intuition by playing with network in the previous sections and trying many other combinations, we came up with the Mice-Mimic-Net given in chapter 5. It has 3 convolutional layers followed by a max polling and another 3 convolutional layers followed with max pooling again, then a fully connected layer of size 1000 and finally a soft-max output layer with 3 neurons, giving one-hot-shot form output. Filter sizes at any layer is chosen as 4x4. Network is trained using Adam Optimizer technique with learning parameter of 0.0001. Since combining pain degrees of each facial element in the training data set gives better accuracy, it was done so. Number of epochs in training procedure is 20k with batch size of 200. Increasing the epoch count further than 20k increases accuracy insignificantly and with no guarantee. Thus 20k training epochs is ideal for obtaining both good accuracy and less training time.

The final training accuracy of the network is 1.0 and test accuracy is around 0.96. It takes about 24 minutes to train the mouse data set on a workstation having a Nvidia Titanx GPU. It is also worth to mention that, trainings are done using Google’s deep learning library TensorFlow [51]. Figure 6.3 gives the accuracies and Figure 6.4 shows the cost (output error - value of cross entropy cost function) while Figure 6.6 visualizes the first convolutional layer kernels and convolutional layer outputs when the input in Figure 6.5 is applied. Note that since weights of the first layer filter either saturates to one or zero, Figure 6.6a seems like a binary image.

Output of the first convolutional layer contains more low level features than the second convolutional layer. Edges of the mouse face are more distinct. On the other hand, after the second convolutional layer, facial elements of the mouse are more distinct, amount of information the frame has decreases. If there were more convolutional layers in the network, higher numbered convolutional layers would have higher level information then the convolutional layers that are close to the input layer. To better illustrate this, 2 more convolutional layers before each max pooling layers in the Mice-Mimic-Net are added and outputs of convolutional layers are given by Figure 6.7. Note the hierarchical information contained at each convolutional layer outputs. The closer the convolutional layer to the output, the higher level information about
mouse image it has.

Figure 6.3: Training and test accuracies. Train and test accuracies are 1 and 0.97 after 20k epochs, respectively.

Figure 6.4: Output error (cost) of the Mice-Mimic-Net goes down to 0.14.
Figure 6.5: Sample test image which is used to visualize convolutional layer outputs.

(a) First convolutional layer filters (32@10x10 and re-sized).

(b) First convolutional layer output.

(c) Second convolutional layer output.

Figure 6.6: Filters and feature maps when the ultimate network is trained based on all the facial elements of the mouse.

6.4 Benchmarking With Classical Methods

6.4.1 Multilayer Perceptron

Multilayer perceptron (MLP) with 2 hidden layer and a soft-max output layer with dropout is trained. Simply, it is a fully connected version of the mice-mimic-net; two convolutional layers are replaced with fully connected layers.

MLP has a poor classification performance compared to convolutional network. The main reason is that, once we connect input pixels via fully connected layer, we loose all the spatial feature of the inputs. Location variance in input images are important in this case. Test and train accuracies when training this network is given in Figure 6.8. Figure 6.9 shows the cross entropy cost at the output of MLP. After 150k epochs,
Figure 6.7: Convolutional layers outputs thorough 1 to 6 when 2 more convolutional layers are inserted before max-pooling layers to the Mice-Mimic-Net. Note the low level to higher level details change from C1 through C6.

network cannot learn anymore. The MLP has about 81 percent train accuracy and 86 percent test accuracy.

Applying PCA reduction on the dataset before MLP classification improves accuracy significantly. When input images are reduced to their principal components which contains 90 percent of the information, 97 percent train accuracy and 95 percent test accuracy is obtained. Accuracy plots of the MLP with PCA reduced images, are given in Figure 6.10.

In general, there is no guarantee that PCA will improve the accuracy of the network. If both of the training and test accuracies of the network is low, then the network is having difficulty to learn. That means one should either try to increase the network parameters (number of neurons and/or layers) or reduce its input size. On the other hand, if the training accuracy is high but the test accuracy is low, that means the network is memorizing (over-fitting to input data). In this case, either networks size should be reduced or input data amount should be increased. In our MLP training, first situation was the case, thus applying PCA on the input data improved the both training and the test accuracies.
Figure 6.8: Accuracies when multilayer perceptron network is used.

Figure 6.9: Cross entropy cost when multilayer perceptron network is used.
6.4.2 Classifying With SVMs

Support vector machines (SVMs) are one of the off-the-shell machine learning techniques. They are explained in Chapter 3.3.1. Table 6.7 shows the training and test performances obtained for different type of SVM kernels. All SVM kernels except the sigmoid kernel, gives perfect training performance, i.e, 1, but only polynomial and linear kernels gives good test performance. Polynomial kernel with gives the best accuracy, which is about 94 percent. When benchmarking with other methods, SVM is assumed to have a polynomial kernel since it has the best performance. When the PCA is applied to the input data, accuracy decreases. This is due to second case explained in section 6.4.1. That is, network performance was good and there was no over-fitting but applying PCA to input, we lost useful information.
Table 6.7: Performance metrics of different SVMs.

<table>
<thead>
<tr>
<th>kernel</th>
<th>train accuracy</th>
<th>test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>linear</td>
<td>1.0</td>
<td>0.894</td>
</tr>
<tr>
<td>polynomial</td>
<td>1.0</td>
<td>0.943</td>
</tr>
<tr>
<td>radial basis</td>
<td>1.0</td>
<td>0.426</td>
</tr>
<tr>
<td>sigmoid</td>
<td>0.423</td>
<td>0.426</td>
</tr>
<tr>
<td>linear with PCA</td>
<td>0.822</td>
<td>0.837</td>
</tr>
<tr>
<td>polynomial with PCA</td>
<td>0.751</td>
<td>0.751</td>
</tr>
</tbody>
</table>

6.4.3 Classifying With K Neighbors Classifier

Table 6.8 shows the results when mouse data set is classified using K-NN with different K values while Table 6.9 shows the results when PCA is applied before classification. K value does not have significant effect on its performance neither PCA pre-processing. When comparing with other classifiers, KNN will be assumed to have 3 as the K value.

Table 6.8: Performance of the K-NN on the mouse data set.

<table>
<thead>
<tr>
<th>K value</th>
<th>train accuracy</th>
<th>test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.977</td>
<td>0.950</td>
</tr>
<tr>
<td>3</td>
<td>0.971</td>
<td>0.945</td>
</tr>
<tr>
<td>4</td>
<td>0.962</td>
<td>0.946</td>
</tr>
<tr>
<td>5</td>
<td>0.956</td>
<td>0.939</td>
</tr>
<tr>
<td>6</td>
<td>0.952</td>
<td>0.941</td>
</tr>
</tbody>
</table>

6.5 Benchmarking All Methods

Figure 6.11 shows classification performance of all the methods. The Mice-Mimic-Net has most accurate train and test performance. Multilayer Perceptron have poor performance when PCA is not applied. This means these networks are having difficulty to classify large input data (under-fitting). On the other hand, PCA does not work for SVM because SVM was already under-fitting to input data. PCA did not have any
Table 6.9: Performance of the K-NN on the mouse data set when PCA is applied to the data set before classification is performed.

<table>
<thead>
<tr>
<th>K value</th>
<th>Train accuracy</th>
<th>Test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.948</td>
<td>0.949</td>
</tr>
<tr>
<td>3</td>
<td>0.97</td>
<td>0.948</td>
</tr>
<tr>
<td>4</td>
<td>0.963</td>
<td>0.944</td>
</tr>
<tr>
<td>5</td>
<td>0.955</td>
<td>0.947</td>
</tr>
<tr>
<td>6</td>
<td>0.952</td>
<td>0.941</td>
</tr>
</tbody>
</table>

effect on the K-NN classifier. This is expected result since both training and test performance of the K-NN is high enough (there is no over-fitting or under-fitting issue). Note that MMN in Figure 6.11 stands for Mice-Mimic-Net.

Figure 6.11: Performance comparison of the methods.
6.5.1 Receiver Operating Characteristics (ROC) Comparisons

ROC curves are widely used for binary classification problem, i.e., for two class classification problems. Area under a ROC curve (AUC) is a measure for performance. ROC curves can also be prepared for 3 or more class classification problem with the "one class vs. all other classes" approach. This is the approach used for comparing the result of different classifiers in this work. Figure 6.12 shows the ROC characteristics of classifiers for the class 0 (pain scale of zero) vs others, Figure 6.13 shows class 1 (pain scale of 1) vs others, and Figure 6.14 shows class 2 (pain scale of two) vs others.

ROC curves gives result correlated with the accuracy comparisons. The Mice-Mimic-Net has the biggest AUC, while K-NN has the second biggest AUC. The bigger the AUC of a classifier’s ROC curve, the better classification performance it has.

![Receiver Operating Characteristic for class 0 vs other two class](image)

Figure 6.12: ROC curve for class 0 (pain scale of 0 - painless) vs other two classes (moderate pain scale and severe pain scale) classification.
6.6 Representivity of the Data Set

Since there are not enough experimental videos of the mice (3), test and training data are both chosen among the same videos. Although training and test sets have totally different frames, since this frames are extracted from the same videos, they have other correlated factors other than pain degree. Both because of this reason and because the data set is small, Mice-Mimic-Net is prone to over-fitting problem. That means, although it performs well when train and test data is chosen this way, it is classification performance is decreased when tried with totally new videos. This problem can be solved with increasing the data set size significantly, which is the later task.

In order to better illustrate this issue, data set is divided into chunks. Then, first 80 percent of each chunk is taken as training data and rest (20 percent) is taken as test
Figure 6.14: ROC curve for class 2 (pain scale of 2 - severe pain) vs other two classes (painless and moderate pain scale) classification.

data. Figure 6.15 shows the performance of the Mice-Mimic-Net when the train and test data is constructed this way. Note that, in the last subplot of Figure 6.15, chunk is a totally new video (biggest chunk). As the frames gets more unrelated spatially, i.e, as the chunk size is increased, classification performance of the network is decreased.

We note that, when the chunk size is small, as in case in above sections, SVM and K-NN have close performance accuracy to Mice-Mimic-Net's performance. As the chunks size is increased, performance gap between them increases. This is because convolutional deep neural networks have better generalization ability.
Figure 6.15: Performances of the classification methods as the chunk size for constructing the train and test data changes. 20 percent of each chunk is chosen as test data.

6.6.1 Using A Completely Different Video As Test Data

Up to now, all the experiments were performed using the data set which is composed of the frames of two different videos. In this section, one of the newly recorded 2 videos is added to train data, while other one is used as test data. In this case, the Mouse-Mimic-Net has given 87 percent test accuracy where SVM with PCA and MLP with PCA produced around 85 percent test accuracies. This results shows that PCA still significantly improves the SVM classifier’s and MLP network’s robustness. This also shows that these networks were still over-fitting since PCA analysis improved their performance. Comparison of the test accuracies and ROC curves are given by Figure 6.16 and Figures 6.17, 6.18, 6.19 respectively. Mice-Mimic-Net is the best classifier since its AUC(Area Under Curve) is highest on the average and SVM is the
worst classifier, and even worse than a random classifier (that has AUC and accuracy of 0.5) since its AUC is lower than 0.5. Note that as the chunk size experiment in the previous sections shows, the more video frames we have (and more kind), the better test accuracy we obtain. Thus, test accuracy of the Mice-Mimic-Net can be further improved by adding more test videos.

Figure 6.16: Performances comparisons of the classification methods when a new video is used as the test data.
Figure 6.17: ROC curve for class 0 (pain scale of 0 - painless) vs other two classes (moderate pain scale and severe pain scale) classification when test data is composed of frames of a completely new video.
Figure 6.18: ROC curve for class 1 (pain scale of 1 - moderate pain) vs other two classes (painless and severe pain scale) classification when test data is composed of frames of a completely new video.
Figure 6.19: ROC curve for class 2 (pain scale of 2 - severe pain) vs other two classes (painless and moderate pain scale) classification when test data is composed of frames of a completely new video.
CHAPTER 7

CONCLUSIONS

Traditional methods of pain recognition and assessment that monitor gross behaviour or clinical signs (for example, weight loss) are time consuming to conduct and limited by the fact that they are not specific to pain. To overcome this issue, FACS (Facial Action Coding System) is designed for facial expression classification in humans. There are many computer based classification methods based on the FACS or similar action coding systems. There are also full automated systems that recognizes the human face, tracks and classifies.

Before the announcement of the Mouse Grimace Scale (MGS) technique by Langford et al at 2010, there were no action coding system for pain scaling in animals. Similar to FACS, Langford et al coded action units of mice for pain scaling. Although MGS technique gave successful classification results, it is a manual method, therefore automation is needed. In this thesis, computational pain classification task on the mice is examined. Mouse faces were labelled into three categories by medical experts as painless (class 0), moderate pain (class 1) and severe pain (class 2) as explained in the MGS manual. Then, a convolutional neural network named as Mice-Mimic-Net is used to classify pain scale of the mouse faces into one of these three classes. The Mice-Mimic-Net performed 96 percent test accuracy, that is, it correctly classified the faces with 96 percent accuracy which are not used to train the Mice-Mimic-Net. On the other hand it performed 86 percent test accuracy for classifying totally a new video. Although this performance is not super good, it is expected that this accuracy can be over 90 percent with more items in the data set. Other classification methods; MLP, SVM, KNN, PCA based SVM and K-NN are also tested on the data.
set but none of them gave the good performance as Mice-Mimic-Net. MPL had the worst performance with the test accuracy of 86 percent, while K-NN the second best classification performance with 93 percent test accuracy. Classification performance of the Mice-Mimic-Net on the mouse data set is very close to state of the art results in the human facial expression classification task, which also utilize deep learning techniques.

Training the Mice-Mimic-Net on a Nvidia Titan-X GPU with 20000 epochs with epoch size 200, takes about 24 minutes while it takes 32.2 hours on a Intel Core\textsuperscript{TM} 2-Duo based CPU training on a personal computer. This shows how GPUs are better than CPUs for mathematics intense computations like neural network training.

For the future work, it would be good if this process is fully automatized. A mouse face detector and tracker needs to be added in order to build detector-tracker-classifier pipeline.
REFERENCES


[34] M. Mufti and A. Khanam. Fuzzy rule based facial expression recognition. In 2006 International Conference on Computational Intelligence for Modelling
Acknowledgments

The authors would like to thank


Glossary

CNN Convolutional Neural Network. 21

deep learning Learning process of deep artificial neural networks. 1

feature map A function which maps a data vector to feature space. In neural network, it maps input features to hidden units to form new features to feed to the next layer. 22

feature vector n-dimensional vector of numerical features that represent some object. 27

filter See kernel. 21, 22

K-NN K-Nearest Neighbors Classifier. 28, 57

kernel Shared weights and biases used for extracting of feature maps. 22, 27, 75

kernel trick Transformation of the feature vectors to a high-dimensional, implicit feature space. Without ever computing the coordinates of the data in that space, one can make calculations computing the inner products between the images of all pairs of data in the feature space. 28

MGS Mouse Grimace Scale. 1

Mice-Net Ultimate convolutional neural network used in this work. 42, 49

mini batch Small number of randomly chosen training samples. 16

normal distribution continuous probability distribution described by the Gaussian normal distribution function. 39

PCA Principal Component Analysis. A method used for dimensionality reduction in a data set. 28
**stride length** slide amount of local receptive field in a convolutional net, in terms of pixel. If receptive field is slid by 3 pixel at each step, then it has stride length of 3. 21

**SVM** Support Vector Machine. 28

**training epoch** A training phase in which all training inputs are consumed once. 17