A MULTI-CAMERA SYSTEM FOR AUTOMATION OF MOUSE GRIMACE SCALING USING CONVOLUTIONAL NEURAL NETWORKS

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

BY

AHMET AĞCA

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

SEPTEMBER 2019

Approval of the thesis:

A MULTI-CAMERA SYSTEM FOR AUTOMATION OF MOUSE GRIMACE SCALING USING CONVOLUTIONAL NEURAL NETWORKS

submitted by **AHMET AĞCA** 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. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Eng.**

Prof. Dr. Uğur Halıcı
Supervisor, **Electrical and Electronics Eng., METU**

Examining Committee Members:

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Eng., METU

Prof. Dr. Uğur Halıcı
Electrical and Electronics Eng., METU

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Eng., METU

Prof. Dr. Turgay Dalkara
Inst. of Neurological Sci. and Psychiatry, Hacettepe Uni.

Assoc. Prof. Dr. Emine Eren Koçak
Inst. of Neurological Sci. and Psychiatry, Hacettepe Uni.

Date: 17.09.2019

ABSTRACT

A MULTI-CAMERA SYSTEM FOR AUTOMATION OF MOUSE GRIMACE SCALING USING CONVOLUTIONAL NEURAL NETWORKS

Ağca, Ahmet
Master of Science, Electrical and Electronics Engineering
Supervisor: Prof. Dr. Uğur Halıcı

September 2019, 127 pages

Over the past decade, convolutional neural networks (CNNs) have gained great progress on the area of computer vision. Many problems related to automation of image recognition or classification are now possible to be solved using CNN with an accuracy much more than a human can achieve. One of these problems is the automation of Mouse Grimace Scaling (MGS). It is such a time consuming and error-prone task even for an expert to classify the pain levels of a mouse for lots of images captured from videos. For this reason, it is essential to incorporate the benefits of popular machine learning algorithms into this research area. The purpose of this thesis is to achieve significant results for practical implementations along with improving the methodology for automation of MGS. In this thesis, a complete set of methodology starting from the mouse monitoring setup to building the neural network model for automation of MGS was studied and the results were compared with that of previous works. For detecting the mouse in video frames, the previously developed tracker algorithms and detection networks were used without change. The evaluation was performed by means of both classification and regression problem and transfer learning was adopted for the basis of the study. For regression, MAE of 0.226 was achieved for one cross-validation (CV) balanced set, and 0.26 was achieved for overall balanced sets (score range, 0 to 2). For binary-classification, 91.10% accuracy was achieved for one CV balanced set, while 82.45% was achieved for overall balanced sets.

Keywords: Mouse Grimace Scaling, Convolutional Neural Network, Transfer Learning, Regression, Automation

ÖZ

EVRİŞİMSEL SİNİR AĞLARI KULLANARAK FARE YÜZ BURUŞTURMASI ÖLÇEKLENDİRME OTOMASYONU İÇİN ÇOKLU KAMERA SİSTEMİ

Ağca, Ahmet
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği
Tez Danışmanı: Prof. Dr. Uğur Halıcı

Eylül 2019, 127 sayfa

Geçtiğimiz 10 yıl içinde, Evrişimsel Sinir Ağları bilgisayarla görü alanında büyük ilerleme kaydetmiştir. Görüntü algılama veya sınıflandırma otomasyonu ile ilgili bir çok problemin çözümü evrişimsel sinir ağları kullanılarak bir insanın başarabileceğinden daha yüksek doğrulukla mümkün olmaktadır. Bu problemlerin bir tanesi de Fare Yüz Buruşturması Ölçeklendirme (Mouse Grimace Scaling, MGS) otomasyonudur. Videodan çıkartılmış bir sürü görüntü için deney faresinin ağrı seviyesini sınıflandırmak bir uzman için bile zaman alıcı ve hataya açık bir işlemdir. Bu nedenle popüler makina öğrenme algoritmalarını bu araştırmaya dahil etmek önem taşımaktadır. Bu tezin amacı MGS otomasyonunu uygulanabilir hale getirmek için kayda değer başarı elde etmenin yanı sıra, otomasyon için kullanılan metodolojiyi de geliştirmektir. Bu tezde, MGS için deney faresini gözlemleme düzeneğinden sinir ağlarını oluşturmaya kadar tam bir otomasyon metodu üzerine çalışılmış ve sonuçlar önceki çalışmaların sonuçlarıyla kıyaslanmıştır. Fare yüzü tespiti için daha önceki bir çalışmada geliştirilmiş fare yüzü takip algoritması ve fare yüzü tespit sinir ağı değiştirilmeden kullanılmıştır. MGS otomasyonu, hem sınıflandırma hem de regresyon problemi olarak ele alınmış ve transfer öğrenimi çalışmanın temelinde yer almıştır. Regresyon için, tek bir eşit ağırlıklandırılmış çapraz doğrulama setinde 0.226

ortalama mutlak hata elde edilmiş, eşit ağırlıklandırılmış çapraz doğrulama ortalaması ise en iyi 0.26 olmuştur (ölçek aralığı 0-2). İkili sınıflandırmada ise, tek bir eşit ağırlıklandırılmış set için %91.10, eşit ağırlıklandırılmış tüm setlerin ortalamasında da %82.45 doğruluk değeri elde edilmiştir.

Anahtar Kelimeler: Fare Yüz Buruşturma Derecelendirme, Evrişimsel Sinir Ağı, Transfer Öğrenimi, Regresyon, Otomasyon

To my dearest wife and beloved parents…

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Prof. Dr. Uğur Halıcı, for her guidance through each step of this thesis study. She always welcomed me whenever I ran into a trouble or had a question about my research. It was a great honor for me to have the opportunity to work with her.

I would also like to thank to the members of Hacettepe University Institute of Neurological Sciences and Psychiatry, Assoc. Prof. Dr. Emine E. Koçak and the graduate students, Canan Ç. Aktaş and Aslıhan B. Varol, for their great collaboration and passionate participation for constructing the dataset.

I would also like to acknowledge my colleague, Ömer Uğur Şahin, for his contribution to the proofreading of all chapters of this work.

By all means, my greatest thanks goes to my spouse, İpek Sarıöz Ağca and my parents, Kamil and Süheyla Ağca, for their continuous support and everlasting belief in me.

Last but not the least, I would like to thank to my company, Aselsan, for providing me free time throughout my master's study.

This study is a continuation of TUBITAK 115E248 - Automatic Evaluation of Pain Related Facial Expressions in Mice (Mice-Mimic) Project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
ANN	Artificial Neural Network
API	Application Program Interface
AU	Action Unit
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Cross-validation
FACS	Facial Action Coding System
FC	Fully-connected
FHD	Full High-Definition
FPS	Frames per Second
GD	Gradient Descent
GPU	Graphical Processor Unit
HU	Hacettepe University
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IN	Isoflurane Anesthesia
INSP	Institute of Neurological Sciences and Psychiatry
KXN	Ketamine/Xylazine Anesthesia
LOSOCV	Leave-one-subject-out Cross Validation
MAE	Mean Absolute Error
ME	Mean Error
MGS	Mouse Grimace Scaling (or Scale)
MLP	Multi-layer Perceptron
MNIST	Modified National Institute of Standards and Technology

MSE	Mean Squared Error
OS	Operating System
PSPI	Prkachin and Solomon Pain Intensity
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
ROI	Region of Interest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
TBI	Traumatic Brain Injury
UNBC	University of Northern British Columbia

CHAPTER 1

INTRODUCTION

1.1. Motivation

Pain has always been one of the most popular subjects throughout the medical history. It is such an essential key factor that it provides clues to the health conditions of living creatures. Besides, the assessment of the pain in the time domain, such as the frequency of pain levels, can give additional medical information. For this reason, it is highly significant to use quantifiable methods for pain level assessments. As in most of the other medical studies, laboratory mouse is usually used in pain related medical researches. In 2010, the first quantifiable method called "Mouse Grimace Scale" (MGS) for scoring the pain level of a laboratory mouse was developed [1]. Despite being a quantifiable method, MGS still requires a highly-experienced coder (expert) in order to achieve an acceptable scoring accuracy. Moreover, the scoring process is a time consuming task and can be considerably subjective under certain conditions. Because of these limitations, it is highly desired to automate the scoring process that will provide cost-effective, fast, reliable, and accurate solution.

In recent years, Convolutional Neural Network (CNN), one type of a deep neural network, has become so much popular in the area of machine learning, specifically in computer vision. In fact, CNNs deserve this popularity because they achieved better results far beyond the expectations for most of the machine learning problems. It is also important to note that the technological advances in integrating the Graphical Processor Units (GPUs) into the network training process made the CNNs feasible for practical implementations. Moreover, one can readily-use the convolutional layers of the popular and successful pre-trained models in order to solve other specific classification problems. This is called the transfer learning method that also significantly shortens the training periods.

In fact, MGS is an image classification or image regression problem. Combining the necessity of automation of MGS and the recent achievements of the CNNs in image classification problems, it is apparent that reliable and accurate results can be obtained for MGS automation using CNNs. Additionally, the results of the previous studies of MGS automation [2]–[5] have already proven the success of the CNNs.

1.2. Contribution

After D. J. Langford *et al.* [1] developed the first grimace scaling for laboratory mouse pain level assessment and published its manual [6], there have been several studies related to grimace scaling, and even for other animals such as rats [7], horses [8], piglets [9] and sheep [10].

Some of the recent studies adopted directly the scoring method of the MGS and tried to automate the process using machine learning. In most of those studies, the evaluation of the network was performed using images taken from different moments. However, in this study, the scoring of the pain level was evaluated over the 6 images captured concurrently from the 6 camcorders placed equiangular in the mouse monitoring setup. Capturing images from different angles of the mouse made it possible to increase the number of training image data, increase the training set reliability and thus increase the accuracy of the network.

The dataset was constructed automatically with the help of previous studies [2], [3], [11], [12] which were carried out within the scope of "Mice-Mimic" project. The tracking algorithm and mouse face detection network were directly adopted from those studies. However, on top of those, a decision algorithm was implemented to combine the results of 6 video files and to select the appropriate frames. Besides, some moments were duplicated in order to be able to analyze the internal MGS consistency of the expert for manual scoring.

In the previous studies, either custom CNNs were used or at most two pre-trained CNNs [4] were included. In our study, in the first phase of training task, the results of 5 pre-trained CNNs were compared, and the one who had the best result was used for the rest of the work.

Although the automation of MGS was addressed as a classification problem in the previous studies, in fact MGS is a regression problem since MGS score calculation [6] results in a continuous value representing how intense the pain level is. Moreover, the behavior of the dependent variables, five Action Units (AUs), can map to a continuous value although they were individually coded in three different groups. When compared multiple MGS scores, a higher value means higher pain level. For ease of application, the resulting scores could also be categorized (discretized) with compromising on accuracy. Thus, the result of a regression network, continuous score value, is much more valuable than that of a classification network. Therefore, in this study, a regression network was developed along with a classification network.

1.3. Organization of the Thesis

In the next chapter, a brief history of facial expression classification and the related studies are introduced. Besides, the summary of the MGS method is given and the milestones in the evaluation of the convolutional neural network are mentioned. A few studies related to the automation of mouse facial expression recognition are addressed with their corresponding performances. At the end of the chapter, the significance of dataset quality for the network performance is emphasized by referring related studies and their results.

In Chapter 3, an overall information of artificial neural networks including most commonly used concepts and the backpropagation algorithm is introduced. Moreover, some specific features of the convolutional neural networks are described.

In Chapter 4, the complete set of methods is explained in detail with a hierarchical way such that they constitute an input-output relationship. The proposed method starts with the mouse experiment and then continues with video processing, dataset construction for manual coding, manual coding, image collection, and dataset construction for training tasks in order. Finally, training and evaluation methods are described.

In Chapter 5, the experimental results of the proposed method described in Chapter 4 are introduced.

In the last chapter, the study is summarized including the method and the experimental results. The results are discussed by means of improvement, possible defects, and comparison with similar studies. The possible future works in order to achieve better results and enhance the methodology are also given.

CHAPTER 2

LITERATURE SURVEY

The potential to be able to classify the expressions of the emotions of man and animals was first stated in Darwin's great study in 1872 [13]. Later, in 1970, Hjortsjö deepened the analysis of the human face that he came up with 23 different muscle groups of specifying facial expressions [14]. Then, his study was taken forward by Paul Ekman and Wallace V. Friesen, and Facial Action Coding System (FACS) [15] was developed in 1978 for classifying human facial expressions. Specifically, for pain intensity assessment, "Prkachin and Solomon Pain Intensity" (PSPI) metric was developed in 2008 on top of FACS.

It is also desired to have such assessment methods for other species for various purposes such as analyzing drug effects and analyzing the correlation between pain levels and neural activity on laboratory animals. For rodents, there have been several methods developed for pain assessment such as conditioned place preference paradigm [16], operant response paradigm [17], self-administration [18], and behavioral-based assessment [19]. Those methods are time-consuming and includes complexity for implementation [20]. Apart from those methods, more recently, quantifying facial expression signatures of a laboratory mouse was also studied. As a result of those efforts, D. J. Langford *et al.* [1] developed the first standardized facial coding scale, MGS, for laboratory mouse pain assessment and published a manual [6] describing the usage of the MGS. The manual basically consists of 3 parts first of which is "Video & Frame Capture" procedure. In this part, the mouse monitoring setup, physical dimensions, and video frame capture method are described. In order to increase the probability of capturing the mouse face, the manual suggested two video cameras for recording at the same time. In the second part, the manual described the coding procedure including the pre-coding process and the five action units (AUs).

The AUs consist of “Orbital Tightening”, “Nose Bulge”, “Cheek Bulge”, “Ear Position”, and “Whisker Change”. Each of the AUs is scored independently with score values 0, 1, and 2 corresponding to the pain levels as Normal, Moderate, and High, respectively. The last part described the calculation and evaluation of the scores acquired in the second part.

The introduction of the MGS provided an acceleration for the grimace studies on other species such as rats, rabbits, horses, piglets, and sheep [7]–[10], [21]. Despite the great contribution of those grimace scales to the medical studies, they all require experts for coding process resulting in a high-cost and a time-consuming task. Moreover, it is not possible to deploy on-line evaluation.

On the other hand, the history of the computer vision dates back to late 1960s when the initial purpose was to build an algorithm to identify the objects in an image [22]. Although there were lots of efforts [23]–[25] for the object recognition of computers, not much progress had been made in a few decades. The studies remained as an ambition rather than a practical implementation until 2001, when Paul Viola and Michael Jones published their studies [26] of real-time object detection using AdaBoost training algorithm. Although the proposed detection algorithm was able to detect several object classes, it was actually used for human face. After that time, various benchmark datasets were published and competitions were organized with those datasets in order to trace the performance of object recognition algorithms over the years. One of those competitions was the “ImageNet Large Scale Visual Recognition Challenge” (ILSVRC) using the subset of ImageNet [27] database as a benchmarking dataset. It is the first time in 2012 that a convolutional neural network showed its real power in the image classification challenge of ILSVRC. Krizhevsky et al. came up with a deep convolutional neural network called “AlexNet” [28] which performed significantly better than its competitors with the top-5 error rate of 15.3% in the image classification tasks. This success was again followed by three CNNs for the following years. VGGNet [29] achieved 7.3% and GoogleNet [30] achieved 6.7%

error rates in 2014 and ResNet [31] achieved 3.57% error rate in 2015 for the image classification tasks.

As the CNNs have gained great improvements in the image classification tasks, they also made it possible to acquire significant results for the recognizing facial expressions. Specifically, for automation of the pain level recognition of humans using facial expressions, several methods had been studied before the rise of CNNs. In a study [32], the training dataset was constructed using the Psychophysiology Laboratory Database of University of Northern British Columbia (UNBC) and the multi-layer perceptron (MLP) was used as a classifier. The paper stated that 91.67% of average accuracy was achieved for the pain/no-pain classification. In another study [33], FACS was adopted as a coding method and 5000 images were trained for a nonlinear "Support Vector Machine" (SVM) classifier. The paper stated that 72% accuracy was achieved for 2-alternative forced choice. For a similar study in 2009 [34], the training dataset was acquired from the UNBC-McMaster shoulder pain expression database and SVM was used as a classifier. The paper stated that 82.4% hit rate was achieved along with 30.1% false acceptance rate for the pain/no-pain classification. In a later work [35], the training dataset was constructed from scratch and FACS was used for coding the images. The paper stated that 81.2% classification accuracy was achieved along with a precision rate of 84% for 2-alternative forced classification. After the CNNs started to gain momentum, the direction of the networks used in pain recognition using human facial expressions changed to the CNN side as well. In 2016, for a different research [36], the source videos in order to generate training dataset were acquired from the UNBC-McMaster shoulder pain expression database. FACS was used to code each image and pain scores are calculated in 16 discrete levels using PSPI method. They handled the pain level recognition as a regression problem and stated that the study achieved average mean squared error of 1.54 for the 16-level PSPI metric. In one of the several similar recent studies [37], 3D CNN was used with a pre-trained 2D architecture. The paper stated that their spatiotemporal CNN called “SCN” achieved average mean squared error of 0.32 for the 16-level PSPI metric.

The studies for the automated mouse pain level recognition using mouse facial expressions are quite limited than that for humans, as the first study for automation of MGS was stated in 2016 [2]. In the study, the dataset was constructed from scratch using the videos of the medical experiments on the mice and the images were labeled by the experienced coders in accordance with the MGS manual [6]. It was stated that 86% test accuracy (3-alternative forced output) was achieved for the samples of a new video [2]. One year later, a similar study was published [5]. In this study, a pre-trained CNN, “InceptionV3” [38] was used with unchanged weights for the convolutional layers and pain/no-pain classification was adopted contrary to its former study. The paper concluded that their network performed 84% accuracy for the validation set and this was increased to 94% if unreliable images were extracted from the validation set. A more recent study, published in 2019 [4], compared 3 CNN architectures, two of which were pre-trained networks, “ResNet50” and “InceptionV3”, and the other was a completely custom CNN. The study adopted MGS as a basis for manual scoring and increased the number of the samples according to the MGS scores according to the time points. The dataset was chosen to be classified as pain/no-pain and thus the last layers of the networks were adjusted to produce a binary classification instead of a regression. Three different types of pain were analyzed independently and the network performances were also evaluated over the averaged results of multiple images of the same moment. The paper concluded that an accuracy of 98.9% was achieved for “ketamine/xylazine anesthesia” (KXN) type of pain using pre-trained ResNet50 CNN architecture with the help of averaging process over multiple images of the same moment. The other results were stated as 89.8% and 90.1% for castration and “isoflurane anesthesia” (IN) types of pain, respectively, using the same network and evaluation method.

For those studies, considering the various types of training datasets, evaluation metrics, precision metrics, labeling methods, and validation methods; it is better to remark that it is almost not possible to make a truly fair comparison between them. This proves the necessity of a regulatory challenge like ILSVRC for mouse pain intensity recognition.

It is also worthwhile to mention the effects of the quality of the training or testing dataset on the accuracy/performance result of an artificial neural network (ANN). Suppose that you have a network of 100% accuracy for the test set classification. If you mislabel the 10% of the dataset, it is obvious that you would get 10% less accuracy. Even this simple example shows that there is a strict correlation between the quality of the dataset and the resulting performance. In the paper published in 2015 [39], the sensitivity of the SVM classifier with respect to the mislabeled training data was studied. It was shown that a decline of 8% accuracy was observed with 20% mislabeled training data. In other words, the accuracy of the classification declined to 82.66% where the original result was 90.66%. Similarly, in a later study [40], the effect of the training label error was studied on a CNN using Modified National Institute of Standards and Technology (MNIST) database. Label error of 4% was randomly injected to the training dataset and the results were compared with the that of untouched training dataset. A decline of around 4%, from 99% to 95%, was observed. Those studies have shown that the reliability of the training dataset should be taken into consideration when evaluating the performance of a neural network.

In this study, the mouse face detection and tracking models were directly used from the previous studies [11], [12]. Since those parts are not the main objective of this work, the corresponding literature reviews were not conducted.

CHAPTER 3

BACKGROUND INFORMATION

3.1. Overview

If the machine is not told how to process the data, it would be totally useless even if you have a super powerful computer of today's technology. Someone has to tell the machine how to process the data. In order to do that, first, the process should be modeled. It is a lot easier for mathematical calculations because they are already modeled processes. It becomes very hard to model the process of identifying an object from an image. On the other hand, it is quite an easy task for a human and even for some other biological creatures. However, it is assumed that the object itself or a similar one was introduced before. This proves the existence of a learning process for biological creatures. In Figure 3.1, a toddler recognizing a giraffe (even it is actually a mock-up) is shown.

Figure 3.1. A Toddler Recognizing Giraffe

We, as humans, are not spending extra effort for the visual recognition tasks. It is something like an intuitive act for us. In fact, this is not the reality. Our brain handles this great job in a silent way. This inspired the researchers that they came up with an artificial neuron model and the rest of the computer vision.

3.2. Artificial Neuron Model

It is estimated that there are about ten billion neurons located in a human brain. The average power dissipation caused by the electrical activity of the overall nervous system is in the order of 10 watts. As shown in Figure 3.2, a biological neuron is composed of three main parts; cell body called soma, axon, and dendrites.

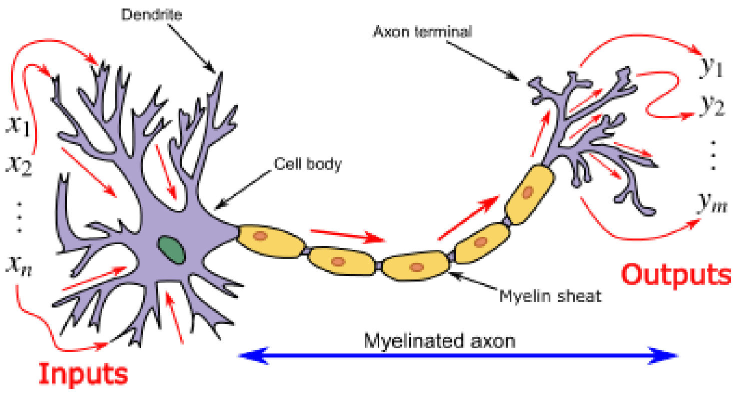


Figure 3.2. Biological to Artificial Neuron [41]

The electrical signal is generated in soma and transferred to other neurons via axon. Axon terminals are connected to other neurons' dendrites with a junction called "synapses". The signal transmission over the synapses is a complex chemical process that causes to raise or lower the electrical potential inside the soma of the receiving neuron. If the electrical potential reaches a threshold, the so-called firing action takes place in the receiving neuron. This process of a biological model was taken as a basis for the first artificial neuron model as shown in Figure 3.3.

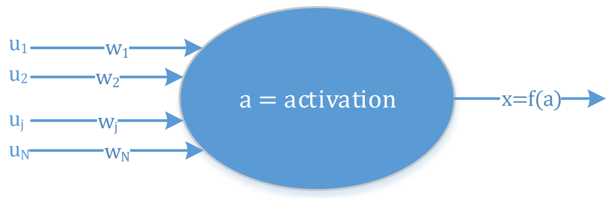


Figure 3.3. Artificial Neuron Model

$$a = (\sum_{j=1}^{N}(u_j w_j) + \theta \qquad (3.1)$$

The activation of an artificial neuron is given in the equation (3.1). "θ" denotes the bias value for the activation. For the ease of use, it is usually included as the 0th element into the summation part. The output of the neuron is actually a function of the activation value. There are several types of activation functions such as; threshold, ramp, sigmoid, Gaussian, Rectified Linear Unit (ReLU), and softmax. The activation function can be included in the hyper-parameters of a neural network, since it effects the performance of the network quite much.

3.3. Neural Network Architectures

Depending on the connection styles of the neurons, different types of architectures are defined.

In feedforward neural networks, the neurons are connected in a hierarchical way that they form layers as shown in Figure 3.4. The neurons in a layer are fed by the previous layer and feed their output to the next layer. In this type of network, connections to the neurons in the same or previous layers are not allowed. The layers are divided into 3 groups. The first layer, input layer, transmits only the applied input to their outputs. The last layer is called the output layer and the layers in between the input layer and the output layer are called hidden layers. If a network consists of only the input and the output layers, then this network is called single layer network. If at least one hidden layer exists, then the network is called multilayer network.

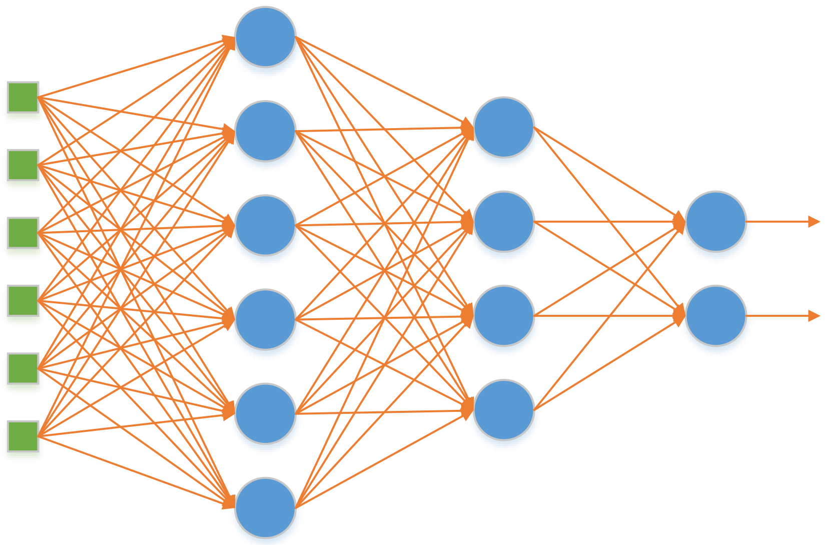


Figure 3.4. Multilayer Feedforward Neural Network

In recurrent neural networks, a connection to the same layer or a previous layer is allowed as shown in Figure 3.5. The idea behind the structure of this network is much closer to biological neuron connections than feedforward neural networks.

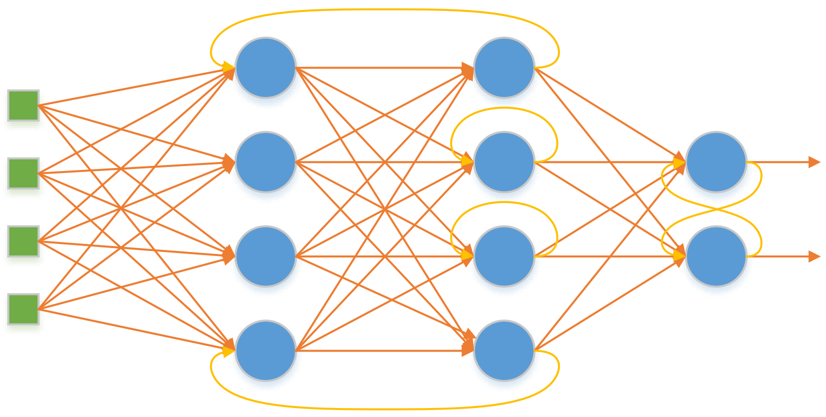


Figure 3.5. Recurrent Neural Network

3.4. Image Classification Using Neural Networks

Let’s think about the relation between the inputs and the outputs of a single layer network shown in Figure 3.6.

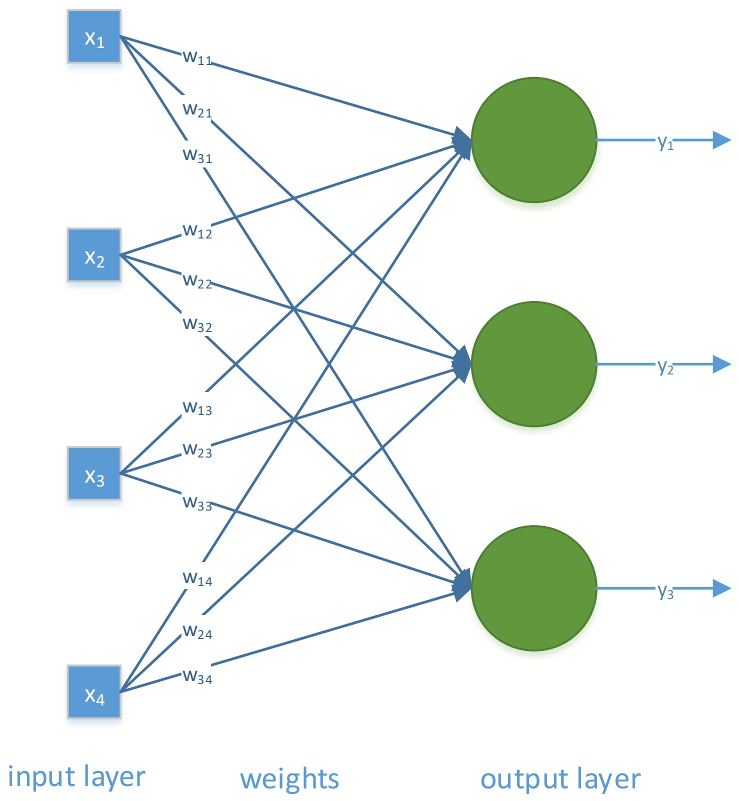


Figure 3.6. Single Layer Network Example

Then, the network can be modeled using the equation given in (3.2) where $\boldsymbol{x}$ denotes the input matrix of 4x1, $\boldsymbol{W}$ denotes the weights as a matrix of 3x4 and $\boldsymbol{y}$ denotes the output matrix of 3x1.

$$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x} \tag{3.2}$$

The same equation can be used to classify an image by applying the pixel values as input and expect the output to be a value corresponding the classification of the image. In fact, the weight matrix is the only one determining the complete relation between the input space and the output space. But, the question here is, what the weights should be. Before that, someone has to tell how good or bad the weight matrix values are. For this, we need to find a suitable metric called a “loss function” or a “cost function”. And, now comes the question, how can we find the perfect combination of weights according to the metric. In fact, this is an optimization problem, since it is not possible to try each and every combination of the weights. And this optimization problem is handled in the training process of the network along with the selected loss function.

3.4.1. Loss Functions

Selecting an appropriate loss function is important, since it is the main actor of the training process and the evaluation of the network model. It is generally a function taking two parameters, the predictions and the desired outputs. The common used loss functions are described in the following sub-sections.

3.4.1.1. Mean Squared Error

Mean Squared Error (MSE) loss is calculated as given in (3.3) that it is basically the average of the squared differences of the ground truth (y_i) and the prediction ($\hat{y}_i$). MSE is generally used for regression outputs.

$$L = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.3}$$

3.4.1.2. Mean Absolute Error

Mean Absolute Error (MAE) loss is calculated as given in (3.4) that it is basically the average of the absolute differences of the ground truth (y_i) and the prediction ($\hat{y}_i$). MAE is generally used for regression outputs.

$$L = \frac{1}{N}\sum_{i=1}^{N} abs(y_i - \hat{y}_i) \tag{3.4}$$

3.4.1.3. Cross-Entropy Loss

Cross entropy is calculated over two distribution vectors, the ground truth distribution and the predicted distribution. Considering only one example of observation, the cross entropy loss function equation would be as in (3.5) where $\boldsymbol{y}$ is the ground truth distribution vector, $\hat{\boldsymbol{y}}$ is the predicted distribution vector, and $\cdot$ is the dot product. If the cross entropy loss function is generalized over all the N observations, the loss function could be rewritten as given in (3.6). For a binary classification problem, the dot product part and write the loss function could be arranged as given in (3.7).

$$L = -\boldsymbol{y} \cdot \log(\hat{\boldsymbol{y}}) \tag{3.5}$$

$$L = -\frac{1}{N}(\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} \boldsymbol{y_i} \cdot \log(\,\hat{\boldsymbol{y}}_{\boldsymbol{i}})) \tag{3.6}$$

$$L = -\frac{1}{N}(\sum_{\boldsymbol{i=1}}^{\boldsymbol{N}} (y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{3.7}$$

The main advantage of this loss function is that it extremely penalizes the case when the prediction is wrong but confident. The cross-entropy loss function and its varieties (binary cross-entropy, categorical cross-entropy, etc.) are generally used after a soft-max layer and they are the most common losses used for classification problems.

3.4.2. Regularization

A general loss function is written with respect to inputs, outputs, and the weights as given in (3.8). This part ensures that the model predictions are close to the ground truth values. However, it is possible to include some other metric to the loss function to penalize some other conditions. For example, it may be desired to have smaller weight values along with ensuring the optimal predictions. In this case, a function of the weight matrix can be included in the loss function as given in (3.9). This part, $\boldsymbol{\lambda} R(\boldsymbol{W})$, is called the regularization term while $\boldsymbol{\lambda}$ is the regularization factor.

$$L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(\boldsymbol{x_i}, \boldsymbol{W}), \boldsymbol{y_i}) \tag{3.8}$$

$$L(\boldsymbol{W}) = \frac{1}{N}\sum_{i=1}^{N} L_i(f(\boldsymbol{x_i}, \boldsymbol{W}), \boldsymbol{y_i}) + \boldsymbol{\lambda} R(\boldsymbol{W}) \tag{3.9}$$

Depending on the function of $R(\boldsymbol{W})$, the regularization is called with different names such as L1, L2, and Elastic net.

There are some other regularization methods specifically used in deep learning. The most common one is the dropout which will be described in deep learning section.

3.4.3. Optimization

The objective of the optimization is nothing but to find the weights that make the loss function minimum for the training process. For a simpler function, f(x), in 1-D, it is much easier to find the minimum in analytical way. Taking the derivative, equating to zero, then checking the points satisfying this equation would probably give the answer. However, for a relatively complex loss function, especially for deeper networks, this analytical approach is not feasible to implement. Another approach to find a local minimum could be starting with an initial point, going towards to one of the sides by checking the derivative of the current point. This is similar to the strategy of a person, with closed eyes, trying to reach the bottom of a sloped land. Since this strategy is useful in practice, it is better to adopt this method for our optimization process. However, since we are dealing in multi-dimensional space, the gradient vector should be calculated rather than the derivative. Then, one can easily calculate the slope with respect to any direction by taking the dot product of the gradient vector with the unit vector of that direction.

3.4.3.1. Gradient Descent

Gradient Descent (GD) is the simplest form of the optimization strategy described above. It is an iterative process of the network training. The weights are updated according to the equation given in (3.10).

$$\boldsymbol{W}_{i+1} = \boldsymbol{W}_i - \eta \boldsymbol{\nabla L}(\boldsymbol{W}_i) \tag{3.10}$$

where η corresponds to the learning rate and $\boldsymbol{\nabla L}(\boldsymbol{W}_i)$ corresponds to the gradient of the loss function at the current weights.

3.4.3.2. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is almost the same as GD that it uses the same formula to update the weights at each iteration. However, the update is executed more frequently because the loss function is calculated over the mini-batches of observations instead of using all observations at once.

3.4.3.3. AdaGrad

AdaGrad is an adaptive gradient method where the learning rate is modified for each parameter. It was first published in 2011 [42]. The learning rate update, given in the equation (3.12), is based on the history of the parameters of the gradient such that the learning rate decays slower for infrequent parameters whereas it decays faster for frequent parameters.

$$\boldsymbol{G} = \sum_{i=1}^{t} (\boldsymbol{\nabla L}(\boldsymbol{W}_i))(\boldsymbol{\nabla L}(\boldsymbol{W}_i))^T \tag{3.11}$$

$$\boldsymbol{W}_{i+1} = \boldsymbol{W}_i - \frac{\eta}{\sqrt{diag(\boldsymbol{G})}} \circ \nabla \boldsymbol{L}(\boldsymbol{W}_i) \tag{3.12}$$

"diag($\boldsymbol{G}$)" corresponds to the sum of the squares of the past gradients of the loss function and ° denotes element-wise multiplication.

Although it is not required to manually adjust the learning rate parameter with the help of AdaGrad optimization, the convergence of the learning rate to zero is a handicap for finding the best solution.

3.4.3.4. RMSProp

RMSProp optimization method tried to eliminate the zero-convergence of learning rate of AdaGrad's method when it was introduced in 2012 [43].

$$\boldsymbol{\vartheta}_i = \gamma \boldsymbol{\vartheta}_{i-1} + (1-\gamma)(\nabla \boldsymbol{L}(\boldsymbol{W}_i))^2 \tag{3.13}$$

$$\boldsymbol{W}_{i+1} = \boldsymbol{W}_i - \frac{\eta}{\sqrt{\boldsymbol{\vartheta}_i}} \nabla \boldsymbol{L}(\boldsymbol{W}_i) \tag{3.14}$$

The technique divides the learning rate by the running average of the magnitude of the previous gradients for each weight. γ corresponds to the decay factor for the running average at each iteration.

3.4.3.5. Adam Optimizer

Adam optimizer, first published in 2014 [44], is one of the most popular optimization methods. It could be considered as an update to the RMSProp method that Adam optimizer uses the estimate of the first moment of the gradient along with the second moment estimate.

$$\boldsymbol{m}_i = \beta_1 \boldsymbol{m}_{i-1} + (1-\beta_1)(\boldsymbol{\nabla L}(\boldsymbol{W}_i)) \tag{3.15}$$

$$\boldsymbol{\vartheta}_i = \beta_2 \boldsymbol{\vartheta}_{i-1} + (1-\beta_2)(\boldsymbol{\nabla L}(\boldsymbol{W}_i))^2 \tag{3.16}$$

$$\widehat{\boldsymbol{m}}_i = \frac{\boldsymbol{m}_i}{(1-\beta_1)^i} \tag{3.17}$$

$$\widehat{\boldsymbol{\vartheta}}_i = \frac{\boldsymbol{\vartheta}_i}{(1-\beta_2)^i} \tag{3.18}$$

$$\boldsymbol{W}_{i+1} = \boldsymbol{W}_i - \frac{\eta \widehat{\boldsymbol{m}}_i}{\sqrt{\boldsymbol{\vartheta}_i + \varepsilon}} \tag{3.19}$$

$\boldsymbol{m}_i$ and $\boldsymbol{\vartheta}_i$ correspond to the running averages of the gradient and the second moment of the gradient, respectively. For the initial iterations, in order to correct the biasing factor, the results of the equations in (3.17) and (3.18) are used for the update of the weights as given in (3.19).

3.4.4. Backpropagation Algorithm

The main objective of the backpropagation algorithm is to find the gradient of the loss function with respect to the weights ($\boldsymbol{\nabla L}(\boldsymbol{W})$ or $\boldsymbol{\nabla}_{\boldsymbol{w}}\boldsymbol{L}$) in order to use in the optimization process. Since there are lots of cascaded connections between the parameters of a network, it is not directly possible to calculate this gradient. Thanks to the chain rule, it made is possible to calculate the gradient of the loss function with respect to each of the weight parameters inside the network.

Let's think of a relatively simple equation given in (3.20) with a computational graph shown in Figure 3.7. It is desired to calculate the partial derivatives of function f with respect to given inputs, at the initial conditions of, $x = 2$, $y = -3$ and $z = 5$.

$$f = x(xy + z) \tag{3.20}$$

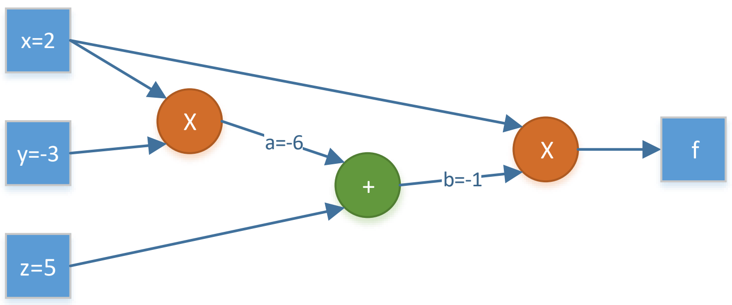


Figure 3.7. Computational Graph of f

$$a = xy, \qquad \frac{\partial a}{\partial x} = y, \qquad \frac{\partial a}{\partial y} = x \tag{3.21}$$

$$b = a + z, \qquad \frac{\partial b}{\partial x} = \frac{\partial b}{\partial a}\frac{\partial a}{\partial x}, \qquad \frac{\partial b}{\partial y} = \frac{\partial b}{\partial a}\frac{\partial a}{\partial y}, \qquad \frac{\partial b}{\partial z} = 1 \tag{3.22}$$

$$f = bx, \qquad \frac{\partial f}{\partial x} = \frac{\partial b}{\partial x}x + b\,, \qquad \frac{\partial f}{\partial y} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial y}, \qquad \frac{\partial f}{\partial z} = \frac{\partial f}{\partial b}\frac{\partial b}{\partial z} \tag{3.23}$$

$$\frac{\partial f}{\partial x} = yx + b = -7\,, \qquad \frac{\partial f}{\partial y} = x^2 = 4, \qquad \frac{\partial f}{\partial z} = x = 5 \tag{3.24}$$

Using the internal derivatives, chain rule and product rule of derivatives, the partial derivatives of f with respect to x, y, and z can be calculated as given in (3.23). And the desired gradients at the given state can be found as given in (3.24).

This method can be generalized for the gradient calculation of the loss function of a multi-layer network. We can summarize the training using backpropagation algorithm in 7 steps as given in Table *3.1*.

Table 3.1. *Backpropagation Training Steps*

Step#	*Description*
1	Initialize weights to random small values.
2	Apply the input, calculate, and store the results of all the layers starting from the input layer to the output layer. (Forward Phase)
3	Calculate the error term using the selected loss function at the output layer.
4	Propagate the error terms for the hidden layers starting from the final hidden layer to the first hidden layer and evaluate the partial derivatives of the errors with respect to the weights at each hidden layer.
5	Combine the results of step 4 in order to get the overall gradient of the loss function with respect to each weight.
6	Update the weights according to the selected optimization algorithm.
7	Repeat steps 2-6 until the stop condition is satisfied.

3.5. Convolutional Neural Networks

3.5.1. Overview

There are two main drawbacks of the classical neural network architecture. The first one is the power of its memorizing capability of the inputs for training. This is caused by the fully connectivity of the neurons between the layers. In other words, the network is not successful to generalize the input space. The other drawback is that each part of the input is treated equally such that spatial structure of the input is ignored. When the problem comes to visual recognition, the spatial structure of the input, usually the image, becomes significant.

The realization of the importance of the spatial structure recognition for the visual system dates back to a study on monkeys in 1968 [45]. The study showed that some specific parts of the monkeys' brains are sensitive when they are shown familiar faces. This fact inspired the existence of the receptive fields for neural networks, and thus the CNNs. In a CNN, the neurons are connected between layers such that they constitute a receptive field when looking from the neuron perspective towards to input.

Although the CNN architecture was not new, a feasible training method had not been developed until 1998, when LeCun et al. implemented the Gradient-based training method for CNN architecture [46]. Their network, LeNet-5, shown in Figure 3.8 was quite successful for recognizing the digits of the zip codes.

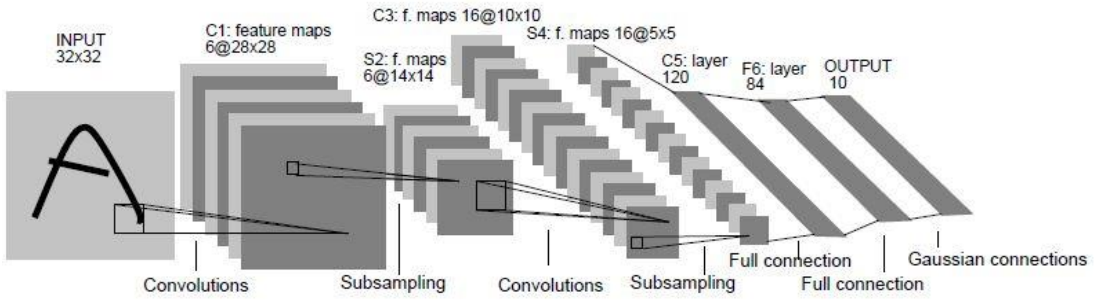


Figure 3.8. LeNet-5 Network Architecture [46]

To the contrary of being successful on digit recognition, the CNNs could not be extended for complex recognition tasks until the introduction of AlexNet [28] in 2012 ILSVRC.

3.5.2. Architecture

Convolution layers have locally connected and shared weight structures like in Figure 3.9. The weights on the arrows of the same color have same values. This provides much smaller number of parameters and local connectivity provides local receptive fields for spatial structure recognition.

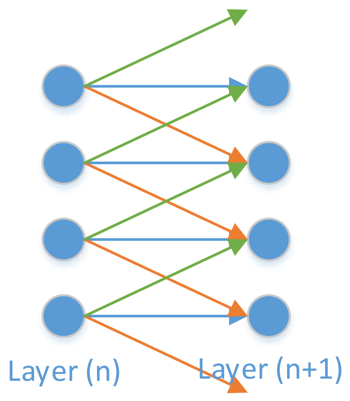


Figure 3.9. CNN Local Connectivity & Shared Weights

The locally connected and shared weights actually behave like a filter (kernel) that results in an output similar to the input but having higher values where the input fragments are similar to the filter, and having lower values where the fragments are not similar.

The filtering feature of a convolution layer is illustrated in Figure 3.10. In the example, the colored pixels correspond to a value 1, where the white pixels correspond to 0. The input is 16x16 pixel image displaying a smiley. The filter (kernel) is formed as 3x3 array such that it tries to find a pattern similar to eye area. Applying the activation function and looking to the resulting image of 14x14 pixels, one can easily point out the locations close to the eye pattern.

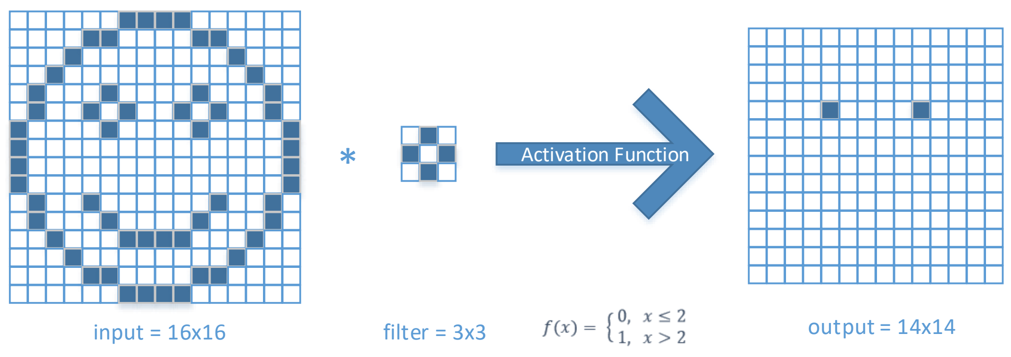

*
Activation Function
input = 16x16
filter = 3x3
$f(x) = \begin{cases} 0, & x \leq 2 \\ 1, & x > 2 \end{cases}$
output = 14x14

Figure 3.10. Convolution Layer Example

As in the example, convolving the filter on the input image is actually sliding the filter over the image spatially while taking the dot product at each position. Thus, only one value is obtained at each position. In this example, the input image has one channel. For the cases of multi-channel input, the convention for 2D convolution is to extend the filter to the depth of the input. That is, if the input size was 16x16x3, then the filter size would extend to 3x3x3. Moreover, multiple filters can be applied in a convolutional layer. For instance, if six filters were applied instead of one for our example shown in Figure 3.10, then the output size would be 14x14x6.

There are some options while executing the convolution operation. These options will be described in the following sub-sections.

3.5.2.1. Stride

Stride defines the step size while sliding the filter over the input. In the example given in Figure 3.10, the stride is chosen as 1. The stride determines the output size of the convolution as given in (3.25). It should be chosen such that the output size results in an integer value.

$$M = \frac{N - F}{S} + 1 \tag{3.25}$$

Assuming all the sizes are equidimensional, M denotes the output size, N denotes the input size, F denotes the filter size and S denotes the stride value.

3.5.2.2. Padding

Padding provides the capability to extend the edges of the input by a desired number of pixels with a value determined by a method. For example, “zero padding” means filling the extended pixels with a value of zero. Including the padding amount to the equation (3.25), results in the equation given in (3.26).

$$M = \frac{N + 2P - F}{S} + 1 \tag{3.26}$$

Assuming all the sizes are equidimensional, M denotes the output size, N denotes the input size, F denotes the filter size, S denotes the stride value and P denotes the padding amount.

3.5.2.3. Pooling

Pooling is actually a layer usually coming after a convolutional layer. It does not hold any parameter in the network model. The aim is to down-sample the size of the resulting activation map of the convolution layer in order to make the network more controllable with fewer parameters. Since it works as a filtering mechanism, “filter size” term is also used along with padding and stride terms. The most common filtering method is “max pooling” where the maximum value is taken within the filtering window. In the “average pooling” method, the average value within the window is taken as the result. The output size of a pooling layer is calculated as given in (3.27).

$$M = \frac{N + 2P - F}{S} + 1 \tag{3.27}$$

Assuming all the sizes are equidimensional, M denotes the output size, N denotes the input size, F denotes the filter size, S denotes the stride value and P denotes the padding amount.

CHAPTER 4

PROPOSED METHOD

4.1. Method Overview

Since the main objective of this study is to come up with a more accurate and reliable solution for the automation of the MGS, building a complete set of methodology that is repeatable, manageable, scalable, modular, feasible, and objective is essential for the automation task and for the future works. For this purpose, a complete set of solutions was studied within the scope of this thesis.

This study was carried out as an extension of the Mice-Mimic project. All but one of the mouse experiments (and the corresponding video recordings) used in this study were conducted within the scope of the Mice-Mimic project before the start of this work. In other words, most of the video files were ready-to-use at the beginning of this study. Two separate Master of Science (MSc) theses [2], [12], were already completed for the Mice-Mimic project. In one of those studies [12], mouse face tracker and detector models were developed for the Mice-Mimic project. Those ANN models were directly used for the video processing task of this study. In the other study of Mice-Mimic project [2], the main objective was same with this study and an ANN called "Mice-Mimic-Net" was developed for the automation of MGS. For the previous study [2], most of the times, it was hard for an expert to score the pain level by looking to a single view of a mouse. It was highly desired to take the advantage of the multi-camera configuration for both manual scoring and prediction tasks. This would minimize the mislabeled data caused by optical-illusion of a single view, in particular. Moreover, evaluating the network results by using multiple frames would increase the prediction accuracy. In this manner, the main extension that was put forward in this study is to use multiple views (of the same moment) of a mouse for both manual scoring and prediction of the pain level. It is also better to indicate that, throughout this study, “moment” term is used to describe a time instant including multiple frames captured by different camcorders.

The block diagram of the overall system is given in Figure 4.1. The process starts with the motivation as described in section 1.1 and ends with the experimental results which are given in Chapter 5. The individual processes of the block diagram are described in detail in the following sections.

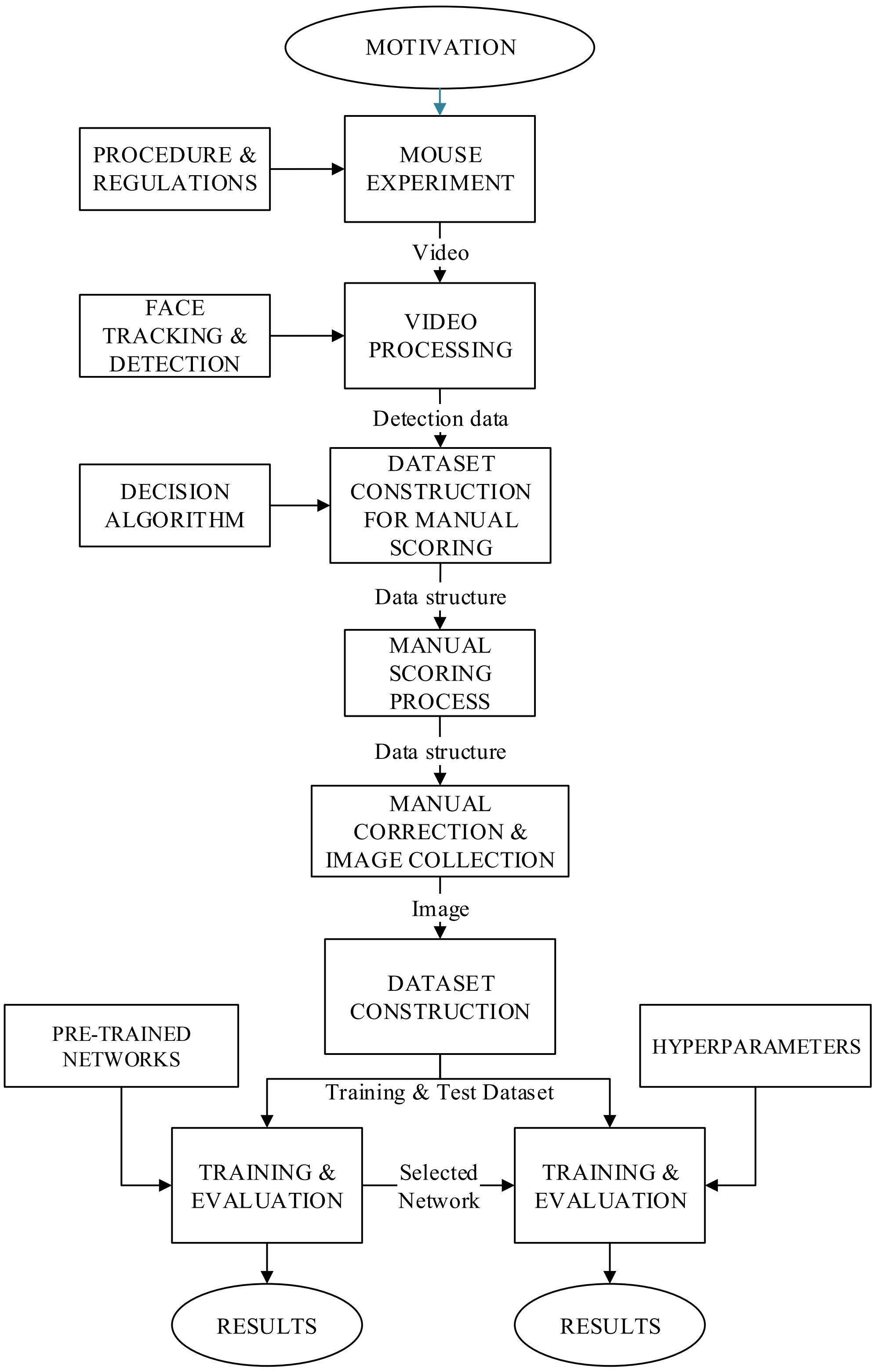


Figure 4.1. Proposed Method Block Diagram

4.2. Mouse Experiment

The purpose of the mouse experiment is to obtain the videos of a mouse which make it possible to analyze the pain levels over a defined time interval. All the experiments were conducted in the Institute of Neurological Sciences and Psychiatry (INSP) of Hacettepe University (HU) laboratories according to the constructed procedures and animal testing regulations. In all experiments, Swiss-Albino mice are used.

For each experiment, two separate recordings are taken for a mouse. The first one, called basal, includes the monitoring of the state of the mouse without pain stimulation and the second one, called pain, includes the monitoring of the pain stimulated mouse.

4.2.1. Experiment Environment

In the previous study of D. J. Langford *et al.*, an experimental environment was proposed with two recorders in their manual [6]. However, this setup has some drawbacks. The most important drawback is the small size of the volume with respect to the size of the animal that may cause undesired stress on the mouse. In order to overcome this problem and to improve the solution approach, a new experimental setup with 6 recorders was developed.

In this new setup shown at the right in Figure 4.2, the mouse is placed in a cylinder with a diameter of 18 cm made of glass. The 6 video recorders are placed around the cylinder at equal angles and the whole setup is enclosed with a hexagonal box made of acrylic glass. The hexagonal box is uniformly illuminated with LED strips in order to standardize the lighting levels among the recordings.

Since combined assessment (for both manual coding process and evaluation of the network) is required for all 6 video recordings, a synchronization method should be used. For this purpose, switching the lighting on/off at the beginning of each recording is included as an additional step in the experiment procedure.

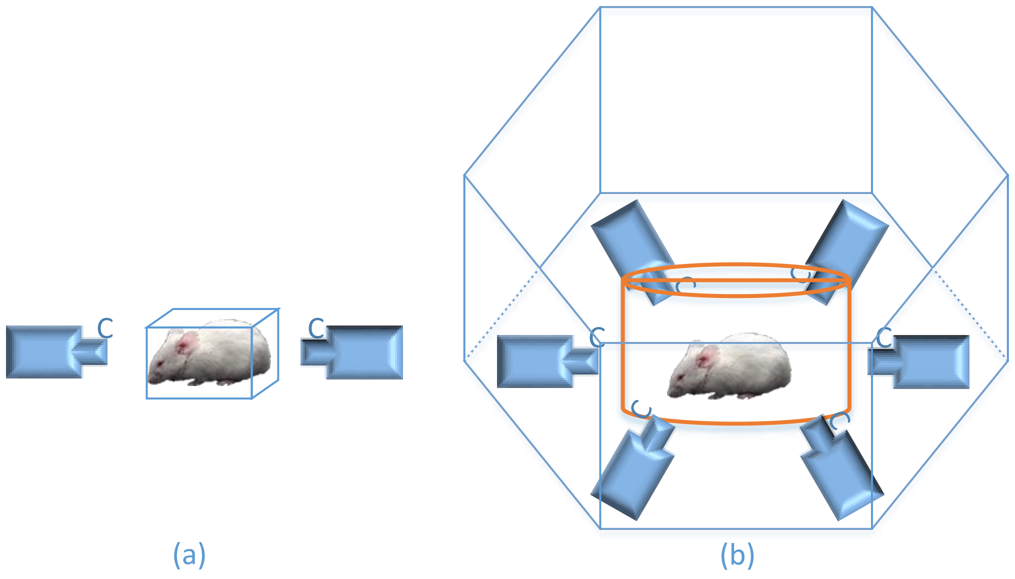


Figure 4.2. Experiment Setup: a) Proposed in [6], b) Used setup in this study

4.2.2. Pain Models

There are several pain models suitable for analyzing the pain intensities of a laboratory mouse. Most of them use chemicals to stimulate the pain as the previous study [1] described some of them. On the other hand, there is another pain model that uses Traumatic Brain Injury (TBI) as pain stimulation.

In this study, only the videos of the experiments conducted according to the traumatic brain injury (TBI) along with their basal videos are used.

4.2.3. Process Output

The output of this process, mouse experiment, is simply the video files. The videos are recorded with 1920x1080 (FHD) resolution, @ 50 fps. For this study, six experiments were conducted with an additional one, constituting a total of seven experiments. There are 2 groups (basal and pain) of 6 video files for each experiment. Thus, a total of 84 video files were obtained as a result of this process.

4.3. Video Processing

The objective of this process is to construct a dataset including mouse face detection data for each video file received from the previous process, mouse experiment. For this purpose, a mouse face detection and tracking model along with an output data structure should be utilized. In the following sub-sections, these are described in detail.

4.3.1. Mouse Face Detection and Tracking Model

Assuming a 60 minute-length FHD video @50 fps, there are 180.000 frames in total. At first, it may be desired to evaluate all video frames for mouse face detection. Considering all the video files, this operation would take much time and in fact not essential. For this reason, one out of every five frames was evaluated starting from the synchronization frame. In order to speed up the process further, a face tracking model was combined with a detection model. The combined models and their related algorithm were directly taken from the previous studies [11], [12], without any change.

The complete algorithm including the combined model and synchronization process is given in Figure 4.3. The initial part of every video file contains a complete dark fragment of a few seconds as the experiment procedure includes a step that the lighting is required to be switched off and on for synchronization purposes. The algorithm starts with the starting frame, then checks the total number of bright pixels with a defined threshold value in the frame. The purpose of this check is to find the synchronization frame of that video file with an accuracy of around 20ms. The checking process is repeated for the next frames until the desired condition is satisfied. After determining the synchronization frame, the algorithm tries to find the detection data of the frames five-by-five. The region of interest (ROI) is easily found by searching the specific color density on the FHD image. Then the region is scanned while the image subarrays are being fed to the face detection network and the detection results are returned. This part (scanning-detection) is usually the most time-consuming part of the algorithm. If the detection is successful, the face tracking network is used for the next frame. The face tracking network feeds the detection network with only one image that is cropped using predicted window. Thus, this process is a lot faster than scanning the ROI. The face tracking network continues to feed the detection network with the next frames until the detection is not successful. In such case, the process continues from the finding ROI part. In the meantime, a simple edge detection filter is applied on the full frame and the number of the pixels belonging to the edges are calculated that provides a simple understanding for the sharpness of the frame.

The results of each processed frame are stored in a defined data structure whether the detection is successful or not. If the frame reaches to the end of the total frames, the algorithm stops.

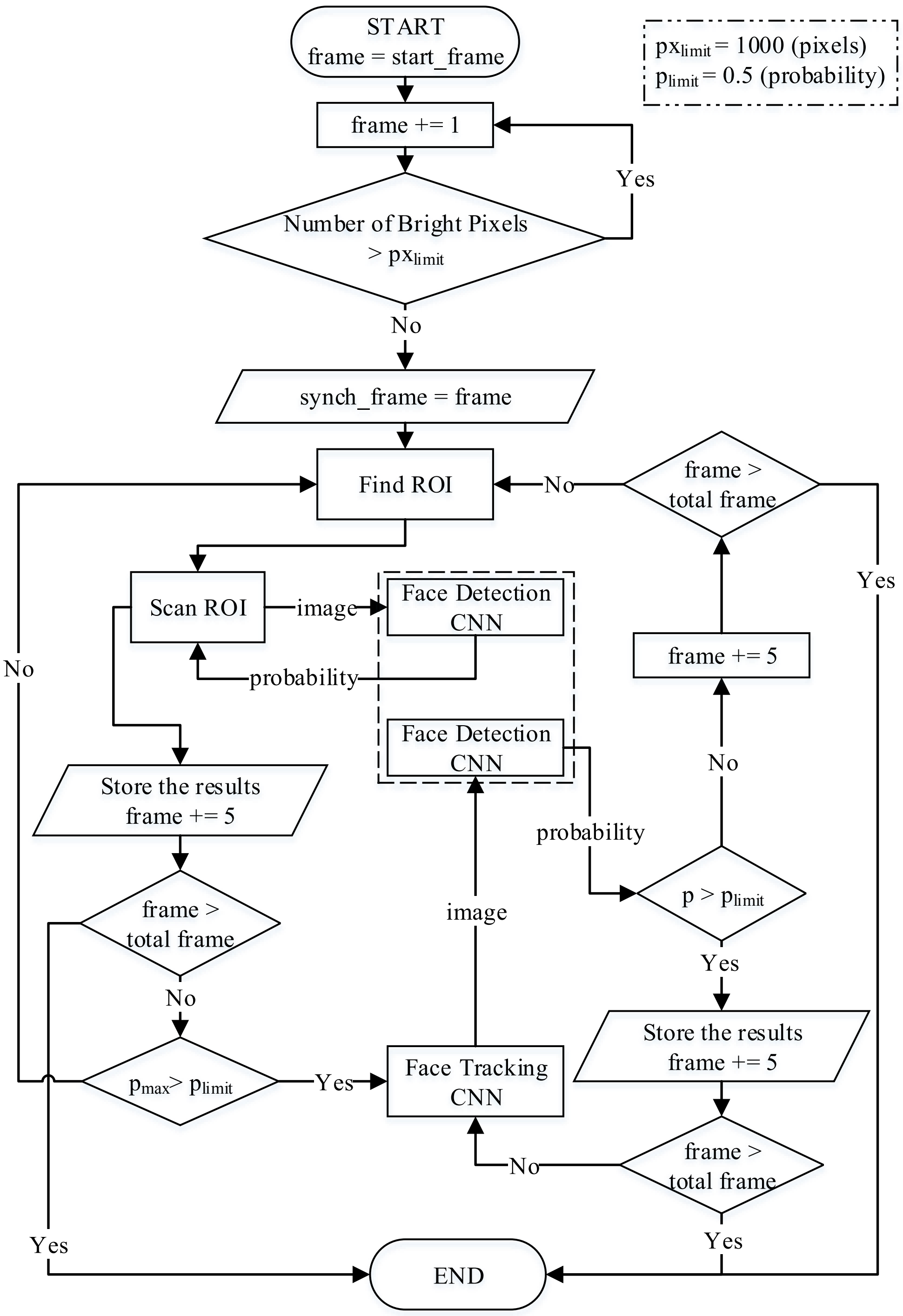


Figure 4.3. Video Processing Algorithm

4.3.2. Process Output

The output of the video processing is nothing but a data whose structure is called "Video Class" and defined in Table *4.1* . The "Video Class" data structure holds all the required information of both the video file itself and the detection information of all the processed frames including the detection box (detection coordinates), detection probability, etc. Thus, the image arrays are not necessarily to be carried within the data structure. In this way, the size of the output data was held at minimum while storing all the information and maximizing the flexibility for the next processes.

Table 4.1. *Video Class Data Structure*

Attribute Name	*Type*	*Description*
dir	String	Full directory of the video file
fr_list	List	List of the objects of Frame Class defined in Table *4.2*
frame_length	Integer	Total Frame Length of the video file
name	String	Name of the video file
path	String	Full path of the video file
save_load_dir	String	Data structure save path
sizeInBytes	Integer	Size of the video file in bytes
sizeInMb	Integer	Size of the video file in megabytes
synch_frame	Integer	Synchronization frame number

Table 4.2. *Frame Class Data Structure*

Attribute Name	*Type*	*Description*
det_box	Ndrray	Numpy array that holds the coordinates of the detection area
det_lbl	Integer	Detection status
det_prb	Float	Detection probability
det_trd	Integer	Tracking detection status
det_trf	Integer	Tracking failure status
frame_num	Integer	Frame number
sharpness	Float	Calculated sharpness level of the image (frame)

4.4. Dataset Construction for Manual Scoring

There are six camcorders monitoring the mouse during an experiment. In Mice-Mimic project, one of the six camcorders was used according to the detection results for scoring process. However, in this study, it is aimed to use all 6 frames for the manual assessment process. This strategy has some advantages. First, the number of data for training and test datasets would increase by including multiple frames for the same moment from different points of view. At this point, it is better to state that the maximum time difference is 20 ms among camcorder frames. Second, the manual scoring accuracy and thus the reliability would increase by presenting the six different views of the mouse to the expert. Moreover, if the overall face expression is desired for one moment rather than the expression from a single view, this method satisfies the needs best. Hence, the coder intuitively scores the average of the suitable views if there are natural minor differences between the views. For some cases, the mouse face expression of the same moment differs much while looking from a different perspective such that sometimes significant differences were observed between the left and right eyes of the mouse.

Considering the conditions stated above, all the data of 6 camcorders received from the previous process should be fused (combined) properly. This operation should also include a selection algorithm because the number of frames are still much even one fifth of the frames were iterated in the previous process. Besides, at some time intervals, the mouse does not move resulting in too much increase at the number of same images which is not desired. For this purpose, an algorithm that fuses and filters the received data from the previous process should be developed.

4.4.1. Fuse and Filter Algorithm

The block diagram of the developed algorithm is given in Figure 4.4. The moments (combined frames of 6 camcorders) are iterated by groups of maximum defined by “max_items_in_det_set” and minimum defined by “min_items_in_det_set”. The maximum limit ensures to select a moment within a maximum defined time. And the minimum limit ensures to select a moment among the minimum number of suitable candidates. During the iteration, if the end of any video frame is reached, the algorithm stops. In the iteration, the combined frame list (moment data) is created using the iteration number (idx) and the synchronization frame number of each video. Then, the average sharpness value is calculated over the detected frames (with a minimum defined detection probability, “min_prob”) of that moment. Comparing the sharpness and number of detections with the previous values of the group, the current moment is decided to be whether the best or not. It is desired to select the moment with the maximum number of detections. The minimum of that number is also defined by “min_det_num”. While maximizing the detection number, it is also desired to select the sharpest moment. For this purpose, the mean of the sharpness, denoted by “sh”, is calculated over the detected frames. The decay factor, “sh_decay”, is included for the case that the number of detections is increased. This is to ensure to accept the moment as best even if “sh” is lower to some extent.

While iterating the moments within the group, the detection boxes of each frames are appended to the multi-dimensional list, “det_box_list”, providing that the detection conditions are satisfied. Then, the mean standard deviation among the detection coordinates is calculated. An example for this calculation is given in Figure 4.5.

If the moment is not found inside the group iteration with the desired conditions stated above, the process is repeated for the next moment providing that the “max_items_in_det_set” is not reached. If the moment is found with the desired conditions, the moment data is stored. If the moment could not be found with the desired conditions and the “max_items_in_det_set” is reached, in this case the best moment (if exists) within the group is stored. Then, the process is repeated for the next group of iteration.

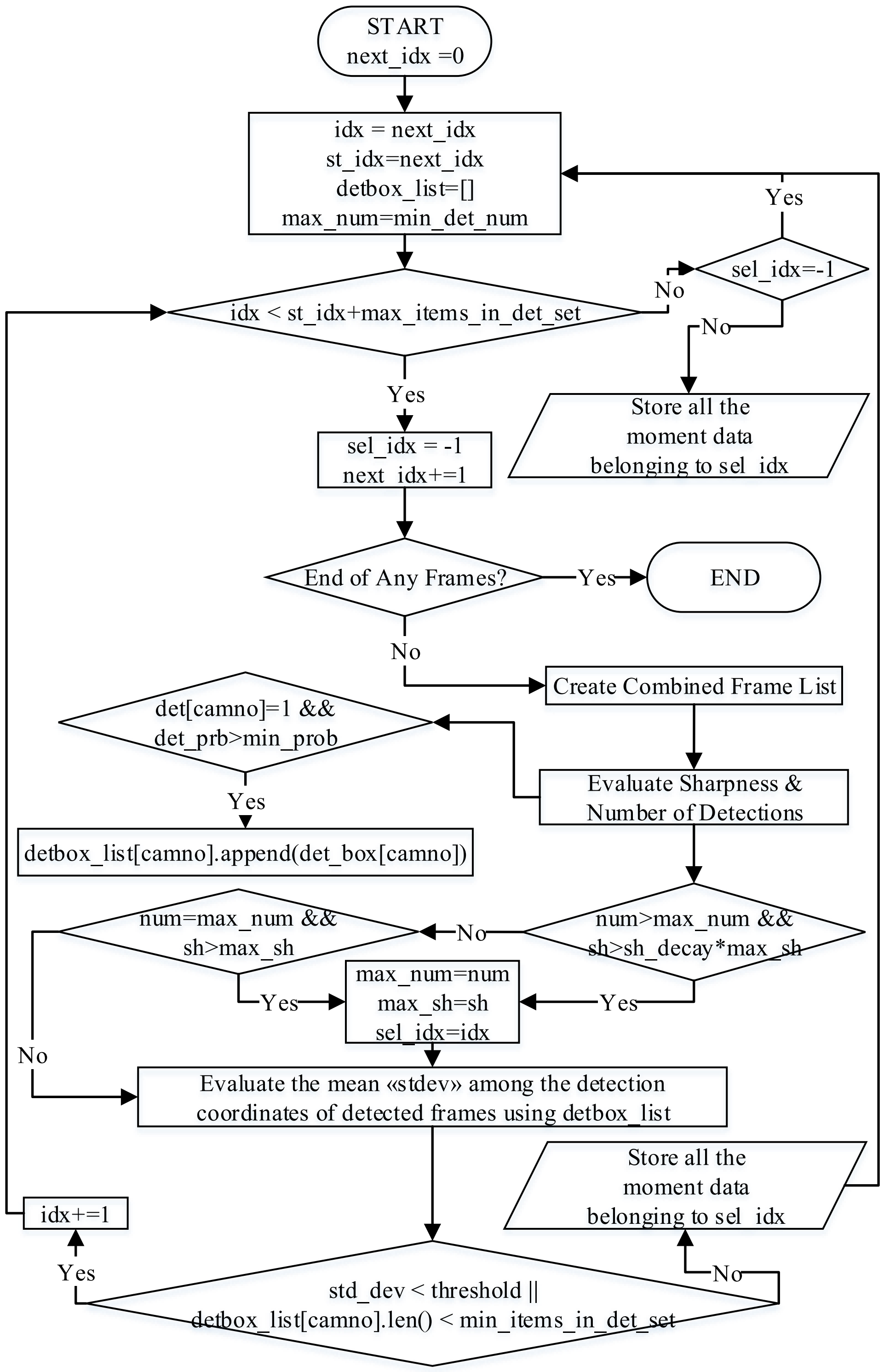
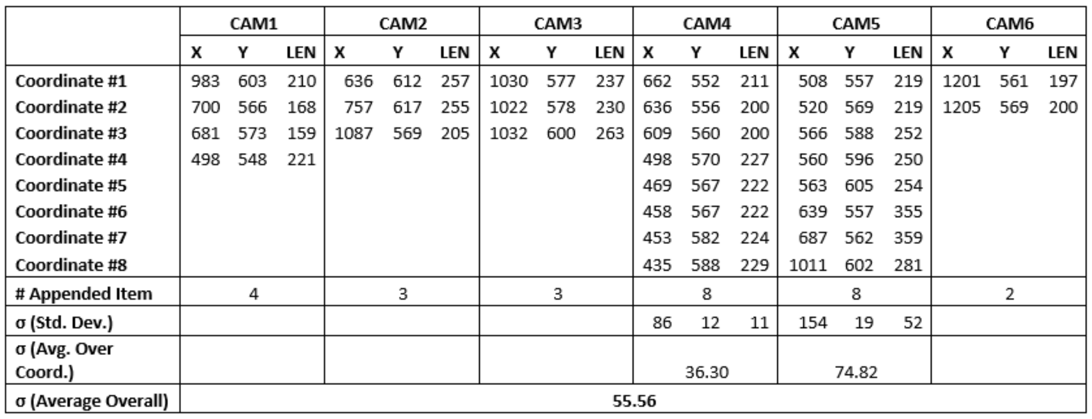

START
next_idx =0
idx = next_idx
st_idx=next_idx
detbox_list=[]
max_num=min_det_num
Yes
sel_idx=-1
No
idx < st_idx+max_items_in_det_set
No
Yes
Store all the moment data belonging to sel_idx
sel_idx = -1
next_idx+=1
End of Any Frames?
Yes
END
No
Create Combined Frame List
det[camno]=1 && det_prb>min_prob
Evaluate Sharpness & Number of Detections
Yes
detbox_list[camno].append(det_box[camno])
num=max_num && sh>max_sh
No
num>max_num && sh>sh_decay*max_sh
max_num=num
max_sh=sh
sel_idx=idx
Yes
Yes
No
Evaluate the mean «stdev» among the detection coordinates of detected frames using detbox_list
Store all the moment data belonging to sel_idx
idx+=1
Yes
No
std_dev < threshold || detbox_list[camno].len() < min_items_in_det_set

Figure 4.4. Fuse and Filter Algorithm



	CAM1			CAM2			CAM3			CAM4			CAM5			CAM6		
	X	Y	LEN	X	Y	LEN	X	Y	LEN	X	Y	LEN	X	Y	LEN	X	Y	LEN
Coordinate #1	983	603	210	636	612	257	1030	577	237	662	552	211	508	557	219	1201	561	197
Coordinate #2	700	566	168	757	617	255	1022	578	230	636	556	200	520	569	219	1205	569	200
Coordinate #3	681	573	159	1087	569	205	1032	600	263	609	560	200	566	588	252			
Coordinate #4	498	548	221							498	570	227	560	596	250			
Coordinate #5										469	567	222	563	605	254			
Coordinate #6										458	567	222	639	557	355			
Coordinate #7										453	582	224	687	562	359			
Coordinate #8										435	588	229	1011	602	281			
# Appended Item	4			3			3			8			8			2		
σ (Std. Dev.)										86	12	11	154	19	52			
σ (Avg. Over Coord.)										36.30			74.82					
σ (Average Overall)	55.56																	

Figure 4.5. Average Standard Deviation Calculation Example

4.4.2. Process Output

The output of this process basically consists of the stored (selected) moments described in the algorithm in section 4.4.1. The number of moments depends on the level of activity of the mouse and the video length with the same parameters in the algorithm. Thus, even though the lengths of the experiments are same, the resulting number of moments could differ among different experiments. These moments are normally listed in time order in the output data structure. However, it is a better idea to present the moments to the expert with randomly shuffled. In this way, the biasing factor for the successive coding of the similar face expressions could be minimized. Another issue is being able to evaluate the internal consistency of a coder. For this purpose, 1 moment is duplicated for every 24 moments in the output dataset, and this information was not shared with the coders. The data structure of the output of this process, called "Experiment Class", is given in Table *4.3*.

Table 4.3. *Experiment Class Data Structure*

Attribute Name	*Type*	*Description*
videos	List	List of the objects of Video Class defined in Table *4.1*
moments	List	List of the objects of Moment Class defined in Table *4.4*
name	String	Given name of the experiment

Table 4.4. *Moment Class Data Structure*

Attribute Name	*Type*	*Description*
Cams	List	List of the objects of Frame Class defined in Table *4.2*
Score	List	List of the objects of Scoring Class defined in Table *4.5*
bad_image	Boolean	Visual status of moment frames for coding

Table 4.5. *Scoring Class Data Structure*

Attribute Name	*Type*	*Description*
Cheek	Integer	Cheek score value
Done	Boolean	Scoring complete flag
Ear	Integer	Ear score value
Eye	Integer	Eye score value
Nose	Integer	Nose score value
Whiskers	Integer	Whiskers score value

4.5. Manual Scoring

Probably the most important part of the whole is the manual scoring process, since the accuracy of the training dataset has a significant role on the performance of the network as described in the previous studies [39], [40]. Moreover, the study in 2012

[47] stated that the global accuracy of a coder with one-year experience was 81% for MGS. The scoring process is also a cumbersome and time-consuming task and requires high concentration even for an expert. This means that it is essential to present a comfortable coding environment using a PC application with a user-friendly interface. For this reason, a PC application called "Label_6Cam" was developed with this perspective.

All the prepared datasets were scored by two experts in INSP of HU. Since, the scoring task was heavy and the coder source was limited, each dataset was manually scored by only one expert.

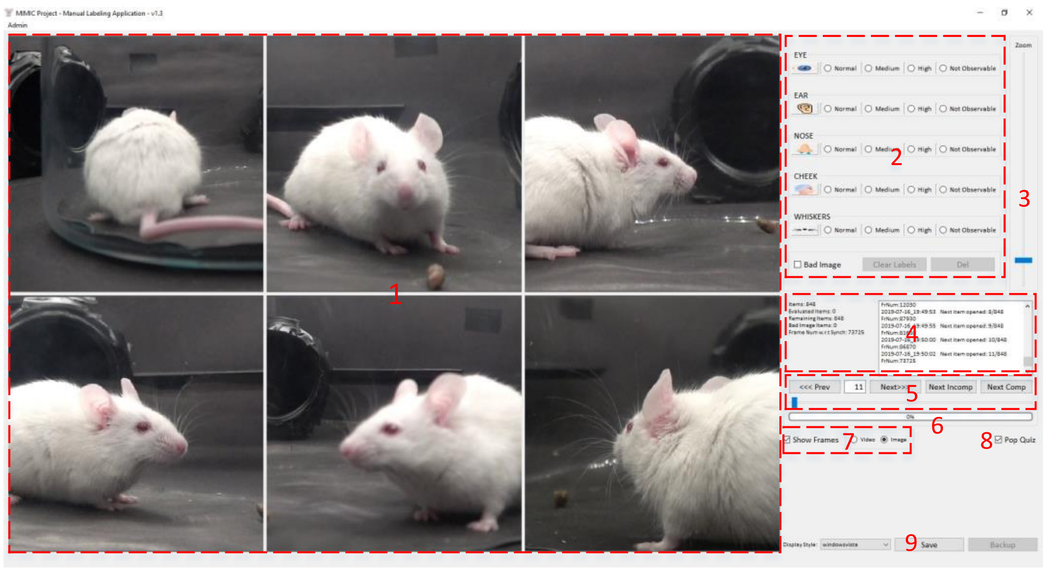
4.5.1. Scoring Application

The general view of the scoring application, Label_6Cam, is given in Figure 4.6 and the description of the interface is given in Table *4.6*.

Python 2.6 is used as the development environment and "pyinstaller" package is used to build the executable (all included package) for Windows Operating System (OS). Thus, the application could run on any Windows machine (XP and later) recommended with an FHD resolution display without the need to install any other software or library.

It only requires the data (.pkl format) received from the previous process to be placed in a subfolder of the application directory and the video path need to be written in the configuration file.

It takes a little longer time while navigating through the items if the frame source is selected as video. In order to overcome this issue, one more feature is added such that all the required images can be captured from the videos and saved in a subfolder automatically with administrative rights. Once the images are captured, the application could use those images for display rather than the videos.



Items: 848
Evaluated Items: 0
Remaining Items: 848
Bad Image Items: 0
Frame Num w.r.t Synch: 73725

```
FrNum:12030
2019-07-16_19:49:53  Next item opened: 8/848
FrNum:87930
2019-07-16_19:49:55  Next item opened: 9/848
FrNum:81
2019-07-16_19:50:00  Next item opened: 10/848
FrNum:86870
2019-07-16_19:50:02  Next item opened: 11/848
FrNum:73725
```

Figure 4.6. Label_6Cam Application

Table 4.6. *Label_6Cam Application Interface Description*

Item#	*Description*
1	Display windows (frames) of the 6 video source
2	Scoring interface
3	Overall zoom option
4	General information area, log of the all actions with timestamps can be reached from this area
5	Navigation tools
6	Coding completion progress bar
7	Display source selection interface
8	Pop-quiz enable option for gathering more attention
9	Manual save button

By default, (at zero zoom level), the parts of the images defined by the detection values are displayed inside the display frames. Using the zoom scroll bar, the coder is able to fit the images to full size.

For scoring (coding) interface, there are 5 facial AUs as specified in [6]. Each of them can be scored separately as Normal, Medium and High. Additionally, the coder can also label the AU as “Not Observable”. If the coder is not comfortable with the images of the moment for scoring, the moment can be labeled as “Bad Image” or left as not evaluated.

One additional feature is the pop-quiz, this feature pops up a window with a trivia question after a certain number of items are coded. The purpose is to gather the attention of the coder while encouraging the coding task.

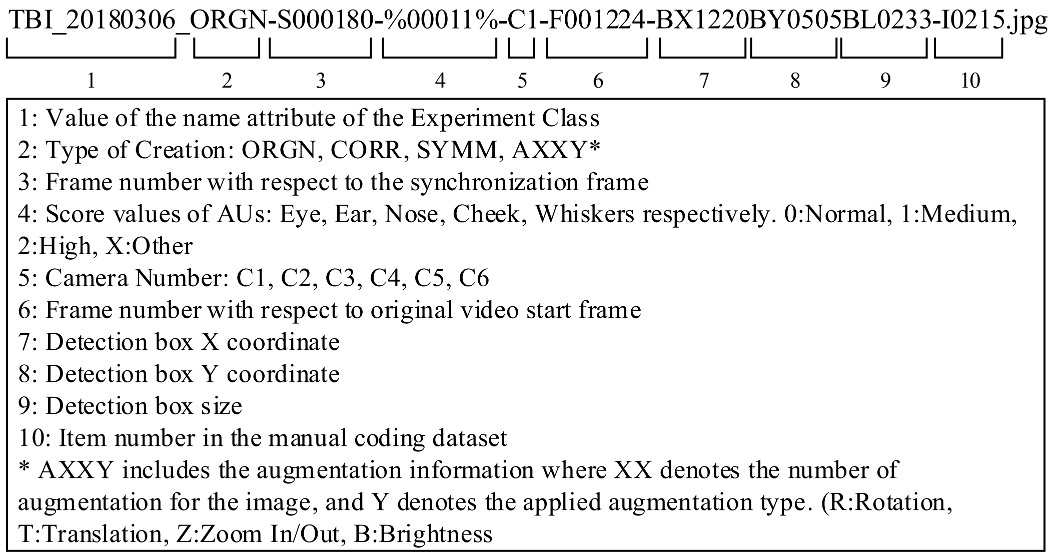
The coder does not need to worry about saving the current state manually, since there is an auto-save option at desired intervals, besides the application automatically saves the last state while it is being closed.

The output of the application has exactly the same data structure with the output of the previous process.

4.6. Manual Correction and Image Collection

Another application that is very similar to the “Label_6Cam” is used for this step, and it is called “Create_Images” application. The scoring interface is disabled; however, it is possible to edit/correct the detection results of the frames. There are two main reasons to edit the detection results of the face detection network. The first one is to increase the number of training data for the situations that the detection algorithm was not able to detect the mouse event though it exists. Secondly, it is desired to decrease the number of unrelated data in the training dataset in order to maximize the quality of the dataset and thus to achieve maximum network performance. This corrected versions of the images were mainly used for training process. However, the results of the uncorrected datasets were considered as the primary result for the evaluation.

Apart from the manual correction feature, the main purpose of the application is to collect the cropped images according to the detection coordinates. The collected images are named with a desired format inside a subfolder. This format can hold almost all the information such that one can regenerate exactly the same image (except for the augmented images) by getting the name and the source video. An example of this naming format and the related descriptions are given in Figure 4.7.


TBI_20180306_ORGN-S000180-%00011%-C1-F001224-BX1220BY0505BL0233-I0215.jpg
1
2
3
4
5
6
7
8
9
10

1: Value of the name attribute of the Experiment Class
2: Type of Creation: ORGN, CORR, SYMM, AXXY*
3: Frame number with respect to the synchronization frame
4: Score values of AUs: Eye, Ear, Nose, Cheek, Whiskers respectively. 0:Normal, 1:Medium, 2:High, X:Other
5: Camera Number: C1, C2, C3, C4, C5, C6
6: Frame number with respect to original video start frame
7: Detection box X coordinate
8: Detection box Y coordinate
9: Detection box size
10: Item number in the manual coding dataset
* AXXY includes the augmentation information where XX denotes the number of augmentation for the image, and Y denotes the applied augmentation type. (R:Rotation, T:Translation, Z:Zoom In/Out, B:Brightness

Figure 4.7. Cropped Image Naming Example

By using such naming format, the output data becomes independent of any process. Any 3rd party could be able to use this data set to train and test their own networks.

The detection boxes calculated from the face tracking and face detection process are too much focused on the mouse face. For this reason, a zoom out operation of 120% is applied while generating original dataset images. Construction of the original dataset results in a single folder with a suffix "_ORGN"; however, the construction of the corrected dataset results in 3 separate folders with suffixes "_CORR", "_SYMM" and "_AUGM". These folders and the number of images that they hold are illustrated in Figure 4.8.

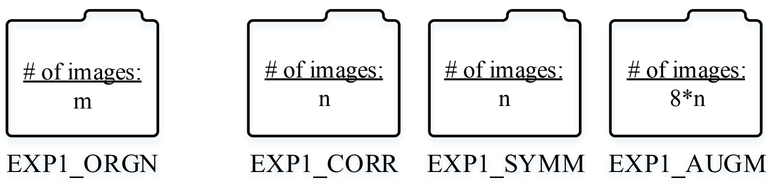


Figure 4.8. An Example of Created Folders of Create Images Application

4.6.1. Data Augmentation

Augmenting data at the image collection step could be the best option since the full image is available with 1920x1080 resolution. Thus, some improved and more realistic methods can be applied such as zooming out and shifting methods using the actual footage. For this purpose, the required interface is also included to the application in order to generate augmented images.

For generating augmented images, first, horizontal symmetry was applied which doubles the number of available data. Then, four types of data augmentation methods were used. These are; rotation, translation, zooming, and brightness change. For each augmentation iteration of each image (with its symmetry form), one of these four methods was chosen randomly with uniform distribution. The iteration number was chosen as 4 which provides an increase to 10 times of the initial number of images as shown in Figure 4.8 and an example of the total 11 generated images is shown in Figure 4.9.

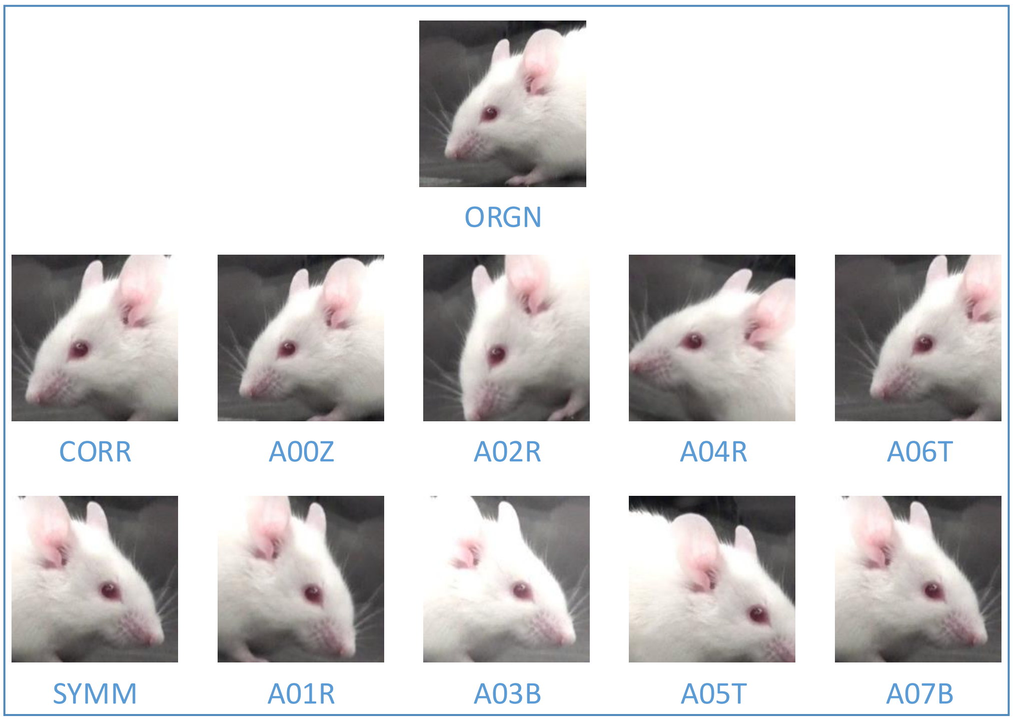


Figure 4.9. Generated Images Example

4.6.1.1. Rotation

For rotation type of augmentation, first, an angle between -45° and 45° was selected randomly with uniform distribution. Then the rotation matrix was calculated and the resulting image was captured using the functions of the OpenCV API. For the cases that the resulting image contains undefined pixels, the repetition of the border pixel method was chosen. An example of this case is shown in Figure 4.10.

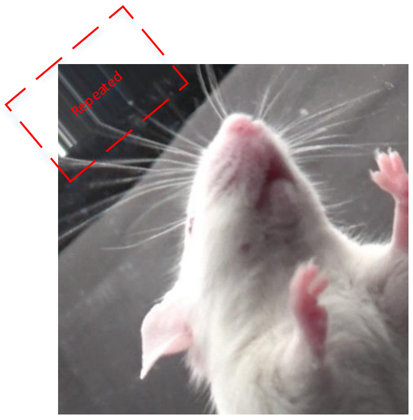


Figure 4.10. Repeated Pixels after Rotation

4.6.1.2. Translation

Translation was applied for both x and y axis of the detected frame. The displacement amount for both axes was chosen randomly from the uniform distribution of numbers between ±25% of image size. For the cases that the chosen linear displacement reaches the outside of the full frame, the displacement was limited to that border of the full frame.

4.6.1.3. Zooming

A zoom amount was chosen from the uniform distribution of the value range of between 75% and 125% of image size. For the cases that the chosen zoom value results in the outside part of the full frame, the zoom amount was limited to the corresponding limit value.

4.6.1.4. Brightness

In order to implement a brightness change to the images, the function called "convertScaleAbs" of the OpenCV API was used. The function takes the input argument "beta" and uses as an offset value for each 8-bit color channel of the pixels of the image. The beta value of the function was chosen randomly from the values between -50 and 50 of a uniform distribution.

4.7. Dataset Construction

The output of the previous process is folders of images that are properly named including all the necessary information. In order to construct the dataset for training and testing, leave-one-subject-out cross validation (LOSOCV) method was adopted in the study. At each iteration of LOSOCV, the dataset belonging to one of the six experiments was excluded from the training dataset. The excluded dataset was used in the evaluation of the iteration.

A separate validation set was also constructed by using the 7th experiment images. Since it is better to use a homogeneously classified (balanced) data also for the validation, some of the other basal experiment images were included in the validation dataset. In the same manner, neither of the validation data was included in any of the training datasets or test datasets.

The images of six different experiments of pain were included to construct LOSOCV datasets. At this point, it is desired to have a homogeneous dataset as much as it could be. For this purpose, the image data (labeled as "Normal" pain intensity) generated from one of the basal experiments was included to all training datasets.

4.7.1. Process Output

The output of this process should include the image arrays and their corresponding score values (labels) at least. A different type of zipped archive file, "npz", was chosen as dataset format. It is generally used as a dictionary-like object including several arrays.

The image files are collected in organized folders as a result of the previous process described in Section 4.6, it may seem not necessary to include the individual image arrays into the "npz" archive, at first. However, the IO operations between the CPU and the disk while feeding the training network could be the bottleneck of the training process. In order to overcome this problem, all the required image data were included into the "npz" archive with the cost of extreme RAM usage.

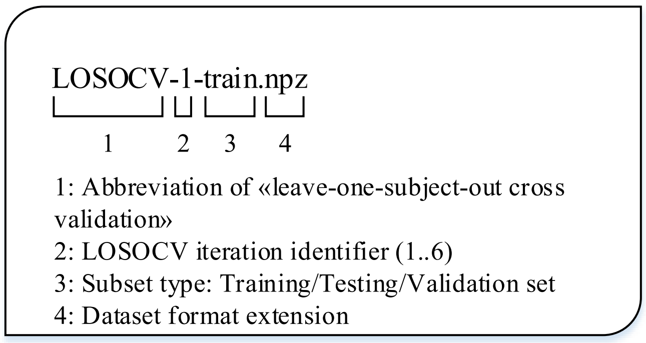
The dictionary-like data structure was used as given in Table *4.7* for the output of this process. And this structure was stored in "npz" format.

Table 4.7. *Dataset NPZ Archive Structure*

Array Name	*Description*
img	Image data array
file_path	Array of relative file paths including the full name of the images
exp_name	Array of concatenation of 1 and 2 values described in Figure 4.7
synch_frame	Array of frame numbers with respect to synchronization frames
score	Array of average score values
score_eye	Array of individual eye scores
score_ear	Array of individual ear scores
score_nose	Array of individual nose scores
score_cheek	Array of individual cheek scores
score_whiskers	Array of individual whiskers scores
camno	Array of corresponding camera numbers
video_frame	Array of frame numbers with respect the start of video frame
box_x	Array of x-coordinates of detection boxes
box_y	Array of x-coordinates of detection boxes
box_length	Array of detection sizes
itemno	Array of item numbers in "Label_6Cam" application

Although most of the arrays included in the dataset archive are not required for any training process, they were included for the sake of data completeness.

Using LOSOCV method requires to generate 6 different groups of datasets each of which includes training, validation, and testing datasets. As described earlier, the validation dataset was prepared using the images of the 7th experiment and some images of one of the basal experiment. The same validation dataset was used for all the LOSOCV training iterations. Since it is useful to see both the corrected data evaluation and the original data evaluation, two separate datasets were constructed for testing. A naming convention was proposed for the constructed datasets, an example of this naming convention is given in Figure 4.11.


LOSOCV-1-train.npz

1 2 3 4

1: Abbreviation of «leave-one-subject-out cross validation»
2: LOSOCV iteration identifier (1..6)
3: Subset type: Training/Testing/Validation set
4: Dataset format extension

Figure 4.11. Dataset Naming Convention Example

In order to decrease the processing time during training operation, the image arrays were also resized according to the network requirements before inserting them into the "npz" archive files. However, the drawback here is that it is required to regenerate the group of datasets for a specific image size requirement.

The contents of the LOSOCV datasets used in the training and the evaluation process are described in Figure 4.12. The different colors represent the different type of datasets constructed in the previous process. The names on the folders correspond to the experiments where "EXP" is a pain experiment and "BAS" is a basal experiment.

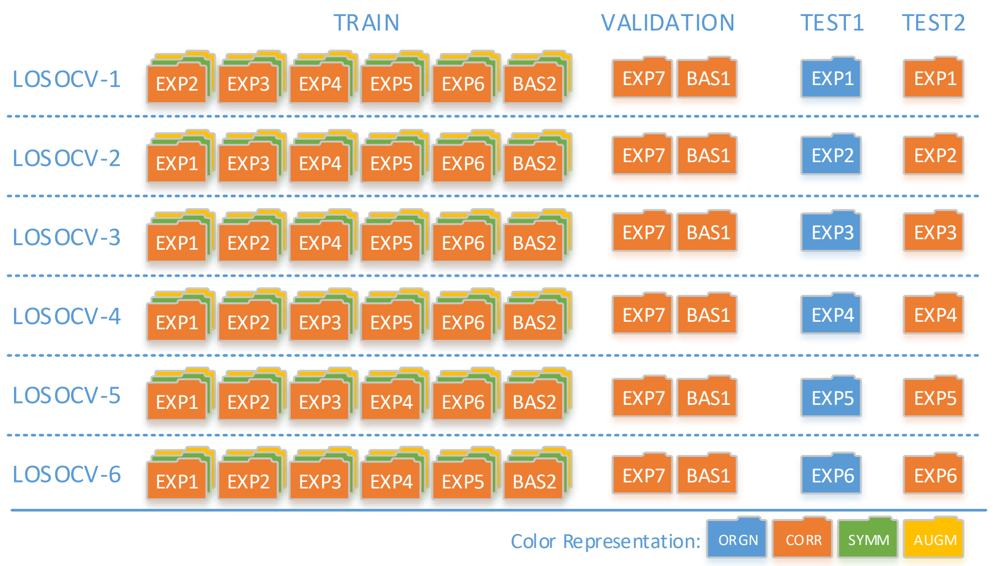


Figure 4.12. LOSOCV Dataset Construction

4.8. Training and Evaluation

In this section, the proposed method for the training and testing process are described. As stated in section 3.5, CNNs are the best network option especially for visual recognition problems. Additionally, with the help of transfer learning, one can take the advantage of pre-trained weights of networks of larger datasets especially for the convolutional layers and by-pass the drawback of the insufficiency of the training dataset and the longer training times. For this purpose, transfer learning method was adopted in this study.

The training process was divided into two phases. In the first phase, the objective was to select 1 out of 5 popular (proved to be successful) pre-trained networks. And in the second phase, the objective was to achieve the best network performance with fine tuning and other applicable methods.

Since, several iterations of training processes are required, it is essential to use a highly manageable, powerful and user-friendly environment. This includes both software and hardware parts. In this study, all the training processes were executed in the cloud.

Working with large datasets requires to train and even evaluate the network using mini-batches, or simply batches. The algorithm/function that feeds the network with batches of data is called “Image Generator”. For this purpose, a custom image generator was developed which will be described in the following sub-sections.

4.8.1. First Phase

The performances of 5 popular pre-trained networks were compared with same and roughly defined hyper-parameters using same datasets. Also, identical fully connected network structure was used after those convolutional layers.

Despite using the pre-trained networks, the training process for one epoch could take longer for a large training dataset of around 250k images. During dry-run training processes, it was observed that even a few training epochs are sufficient to get the best validation result. For this reason, it was decided to use a callback function that provides early stopping the training process for the condition that the validation set evaluation does not improve in a definite number of epochs.

The most of the work will be analyzed within the scope of the regression output. However, the comparison of classification performances of these networks are also included in the study.

4.8.2. Second Phase

With the selected convolutional neural network in the first phase, it is desired to achieve the best performance using alternative methods. One of them is the selection of hyper-parameters such as learning rate and activation function. The other one could be applying normalization after fully connected layers or before input layer.

The fully connected network structure could be another key factor and thus different structures of fully connected layers are studied.

The last option is to train the convolution layers with a lower learning rate along with the fully connected ones.

The binary (pain/no-pain) classification performances of the network are also included in this final phase. Having the advantage of the multi-camera system, the final evaluation also includes multi-camera based (multiple frames of the same moment) evaluation results.

4.8.3. Image Generator

The main purpose of an image generator is to feed the network in small groups of data during the training or evaluation process. This necessity actually rises from the limited sources of the hardware. For large datasets, it may not be possible to load the required data at once even for the initial processing. Considering the training process, trying to include all the training data at once would not be a feasible approach. For this reason, image generators are essential, not an option, for large dataset training processes with limited hardware sources. However, still there are other problems related to IO operations while trying to massively read individual image files from the disks. Although some alternative methods exist such as multiprocessing, the fastest and most reliable way is to directly feed the data to the training process from RAM content. For this purpose and having the advantage of cloud computing, virtual machines with higher RAM capacities were used in order to make the image generator possible to read any desired data in the fastest way.

Image generators can also have side functions such as shuffling dataset, balancing dataset, and augmenting data. For the test or validation datasets, those functions may not be required except for balancing dataset. Thus, in this study, two different image generators were developed one of which is capable of shuffling and balancing dataset (called IMGEN1) and the other does not have any auxiliary function (called IMGEN2). The function of the latter is to split the dataset into small chunks with defined batch size and feed the related process with those chunks. Similarly, the function of the former is also to create mini-batches. In addition to the main function, batches can be arranged in such a balanced way that there would be equal number of data from different score ranges (classes). For the cases that the number of data differs among classes, some of the data are repeated inside the classes in which there are less data. Moreover, IMGEN1 is capable of shuffling at two different steps. First, the complete dataset can be shuffled initially (at the start of each epoch) and then, the created mini-batches can also be shuffled (at the end of each batches) just before feeding the process. These two shuffling methods ensure the complete randomness especially for the training network.

The algorithm of the IMGEN1 is given in Figure 4.13. And a simple example of mini-batch creation of IMGEN1 is given in Figure 4.14.

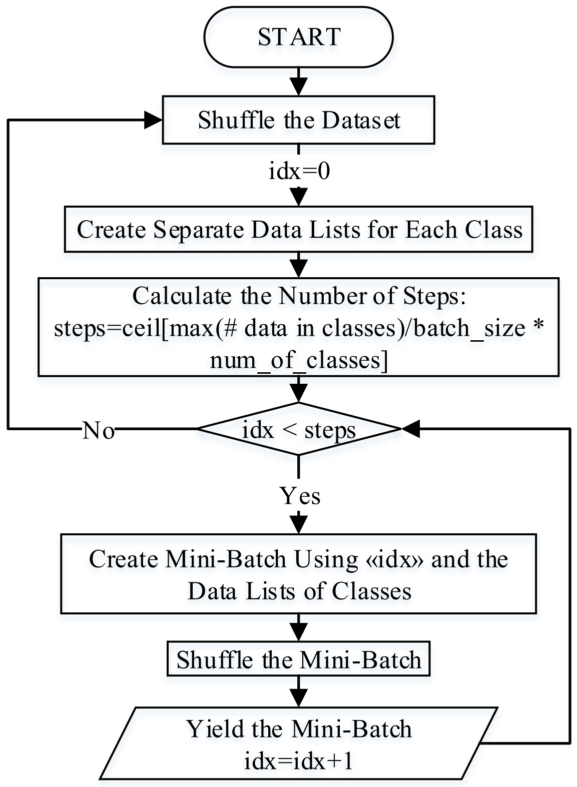

START
Shuffle the Dataset
idx=0
Create Separate Data Lists for Each Class
Calculate the Number of Steps:
steps=ceil[max(# data in classes)/batch_size * num_of_classes]
No
idx < steps
Yes
Create Mini-Batch Using «idx» and the Data Lists of Classes
Shuffle the Mini-Batch
Yield the Mini-Batch
idx=idx+1

Figure 4.13. Train Image Generator Algorithm

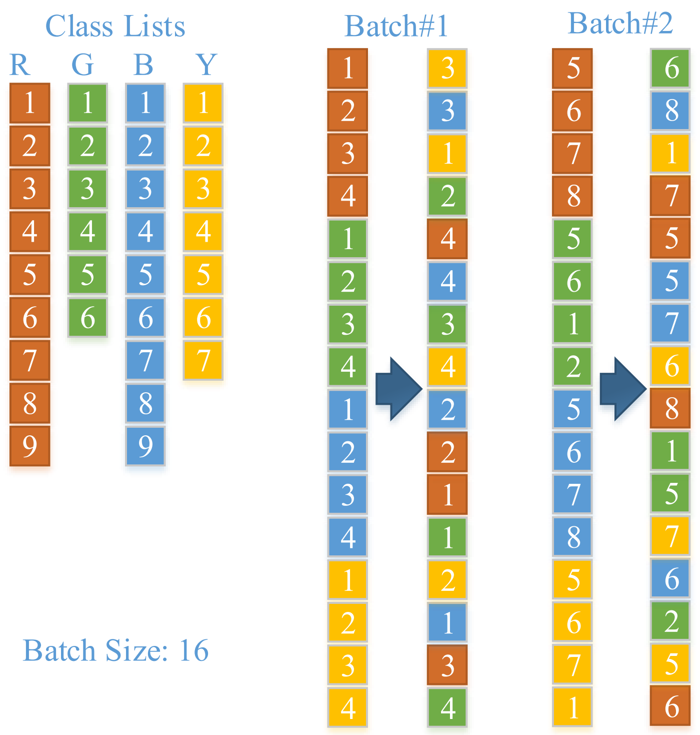


Figure 4.14. Mini-Batch Creation Example

4.8.4. Evaluation Criteria

For classification type output, the selection of the evaluation criteria is quite straightforward. It is the “accuracy” that is defined as the fraction of the successfully classified inputs to the total number of inputs. Although the performance evaluation metric of the classification type network was the “accuracy”, the metric chosen for the early stopping callback was either the “loss function” or the “accuracy” in this study.

When it comes to regression type of networks, it is not meaningful to use the accuracy as the performance criteria, instead the loss function can be used. MSE and MAE are the most common loss functions for regression outputs. If the square domain is not desired, Root Mean Square Error (RMSE) can also be used as an alternative of MSE function. Selecting one of them was not an easy choice in this study since there is no correct answer for that in general. MSE/RMSE is useful if we are more interested in the less frequent unexpected values. Thus, the drawback is that the error gets worse even if there is a single very bad prediction. In other words, this method is sensitive to the less frequent large errors. On the other hand, MAE gives an intuition on how the network performs, easily. This is because it behaves linearly to each of the error values. It can be stated that MAE is not that sensitive to the individual large errors as MSE/RMSE does.

In some previous studies [48], [49], MSE/RMSE was not recommended as an evaluation metric for the average model performance. It was claimed that RMSE adds an ambiguity to the average error evaluation process since it is dependent on the three different characteristics of the errors rather than one. On the other hand, in a later study [50], it was suggested that using RMSE as an evaluation criteria is more suitable than MAE if the error distribution of the system is close to Gaussian.

Combining all the information above, in this study, it was decided to choose the loss function by analyzing the error distribution of manual coding process (duplicated moments were presented to the experts without they were informed as it was explained in Section 4.4). If the manual coding error distribution reflects a similar distribution to a Gaussian one, it would be better to choose the MSE or RMSE rather than MAE. However, a question arises that how the error distribution could be compared to a Gaussian one. It is known that the maximum value of $\left(\frac{RMSE}{MAE}\right)$ is equal to 1, where the error is distributed uniformly. Perturbing this uniform distribution causes an increase in the ratio of $\left(\frac{RMSE}{MAE}\right)$. The ratio converges to a fixed value of $(\frac{1}{0.8})$, which is 1.25, if the distribution is Gaussian as stated in the previous study [50]. Perturbing the Gaussian distribution by adding outliners causes a further increase in the ratio. Thus, the ratio can give an understanding on how much outliners exist in a distribution with respect to a Gaussian or uniform distribution.

Considering the facts stated above, a threshold value for $\left(\frac{RMSE}{MAE}\right)$ of manual coding errors is determined using that of the Gaussian distribution with a tolerance of +10%, which equals to 1.375. If the ratio, $\left(\frac{RMSE}{MAE}\right)$, calculated from the manual coding errors is less than 1.375, MSE or RMSE will be selected, otherwise MAE will be used.

4.8.5. Software

There are lots of frameworks available for machine learning development such as TensorFlow, Caffe2, PyTorch, MXNet, CNTK. The additional software and their versions are generally dependent on the selected framework. In this study, TensorFlow framework was used to build the network and the other software information is given in Table *4.8*.

Table 4.8. *Software Configuration*

Name	*Version*	*Description*
Python	3.6.8	Programming Language Environment
TensorFlow	1.14.0	Machine Learning Framework
Keras	NA	Machine Learning High Level API for Tensorflow 1.14
NumPy	1.16.4	Scientific Computing Package
Opencv-Python	4.1.0	Python Library for Computer Vision

4.8.6. Hardware

Training large networks with large datasets is a time-consuming task for a standard CPU. Higher capacity RAMs are also required. In order to decrease the computation time, GPU support can be used within the limitations of the selected framework.

Since, several training runs will be executed in this study, it is highly required to have a powerful hardware configuration, and even multiple instances of it if possible. For this purpose, Google Cloud Platform was used for all the training processes. Multiple virtual machine instances were used with the hardware configuration given in Table *4.9*.

Table 4.9. *VM Hardware Configuration*

Name	*Description*
CPU	2 x vCPU (3.5 GHz, Single-Core Max Turbo frequency)
RAM	52 GB
GPU	NVIDIA Tesla P100 (16 GB HBM2 RAM)
HDD	30 GB
SSD	375 GB (Shared among VM instances)

Since there are multiple instances running for a common purpose, they should be managed in a centralized way such that they should share the common sources like datasets and executable codes. Besides, the outputs of the VM instances should be saved in a common data storage. For this purpose, Google Storage Buckets were used. The main advantage of this is that one single storage bucket can be mounted to multiple VM instances. Similarly, read only disks can also be mounted to multiple instances. By this way, the input and output data could be easily managed among the compute engine instances. Furthermore, data transfers between the Storage Buckets and Google Drive could also be easily achieved through the help of Google Colaboratory environment with a few line of codes. These connectivity features of different types of tools in the cloud made them appealing to be used in this study. These connectivity features are illustrated in a block diagram given in Figure 4.15.

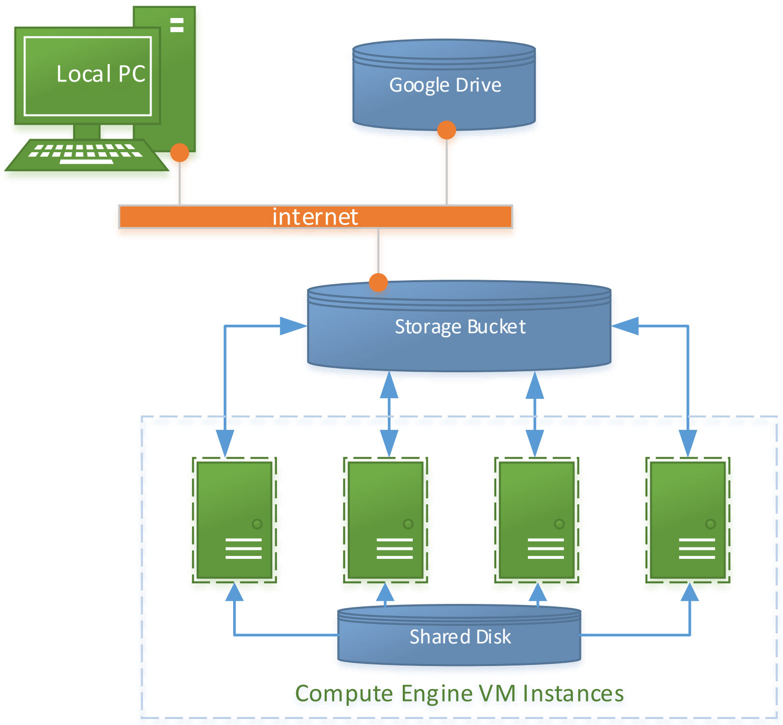


Figure 4.15. Cloud Tools Connectivity (High Volume Data Sharing)

CHAPTER 5

EXPERIMENTAL RESULTS

5.1. Mouse Experiment Outputs

In this study, the videos collected in six experiments with an additional one experiment were used in order to construct a validation dataset. There are 2 groups (basal and pain) of 6 video files for each experiment. However, only 2 basal parts of the experiments were used. The durations and the total frame numbers of the video files used in this study are given in Table *5.1*. All the videos were recorded in FHD resolution @50 fps.

Table 5.1. *Collected Video File Durations in Minutes (Frames in Parenthesis)*

Name	*CAM1*	*CAM2*	*CAM3*	*CAM4*	*CAM5*	*CAM6*
EXP1	94.30 (282912)	94.30 (282912)	94.40 (283200)	94.46 (283392)	94.52 (283560)	94.46 (283392)
EXP2	65.74 (197232)	65.82 (197472)	65.77 (197304)	65.78 (197352)	65.79 (197376)	65.79 (197376)
EXP3	65.54 (196608)	65.57 (196704)	65.62 (196848)	65.62 (196848)	65.62 (196848)	65.59 (196776)
EXP4	65.66 (196992)	65.60 (196800)	65.72 (197160)	65.61 (196824)	65.60 (196800)	65.50 (196488)
EXP5	68.07 (204216)	67.99 (203976)	68.01 (204024)	68.12 (204360)	68.07 (204216)	68.08 (204240)
EXP6	77.34 (232032)	77.31 (231936)	77.31 (231936)	77.30 (231912)	77.30 (231912)	77.26 (231792)
EXP7	61.22 (183672)	61.17 (183504)	61.17 (183504)	61.19 (183576)	61.10 (183312)	61.18 (183528)
BAS1	45.42 (136248)	45.34 (136032)	45.46 (136392)	45.69 (137064)	45.62 (136872)	45.43 (136296)
BAS2	28.36 (85080)	28.22 (84672)	28.30 (84912)	28.36 (85080)	28.37 (85104)	28.37 (85104)

5.2. Manual Coding Dataset

A total of 7 different datasets were constructed for manual coding of pain experiments and 2 different datasets were constructed for manual coding of basal experiments. The algorithm described in section 4.4.1 was used to construct each dataset with the parameters given in Table 5.2.

Table 5.2. *Fuse and Filter Algorithm Parameter Values*

Parameter	*Value (Pain)*	*Value (Basal)*
min_det_num	3	3
min_items_in_det_set	8	20(BAS1), 16(BAS2)
max_items_in_det_set	100	100
min_prob	0.95	0.95
threshold	12	12
sh_decay	0.9	0.9

The resulting number of moments selected for each dataset and the corresponding scorers are given in Table 5.3.

Table 5.3. *Number of Selected Moments for Manual Scoring*

Name	*Number of Moments*	*Scorer(s)*
EXP1	1031	Expert#1
EXP2	911	Expert#1
EXP3	728	Expert#1
EXP4	848	Expert#1 and Expert#2
EXP5	598	Expert#2
EXP6	1002	Expert#2
EXP7	468	Expert#2
BAS1	956	Expert#1
BAS2	818	Expert#2
TOTAL	7360	

A selected moment can be labeled as "Bad" if the expert considers that the quality of the frames captured for that moment is insufficient for manual scoring. In this case, the moment data was ignored for the next process. The expert can also label an AU as

"Not Observable". If there are more than two AUs that are not labeled or labeled as "Not Observable", the corresponding moment data was ignored for the next process as in the case of "Bad" moments. Those moments are called "unaccepted" moments. In addition to those, since 4% of the moments were duplicated inside the dataset for the consistency analysis, one moment of each duplication was also ignored. In the next process, it will be possible to modify mouse face detection values (detection status, detection coordinates). In order to prevent any ambiguity, the dataset without the manual correction is called "original" (ORGN) and the same dataset with applied manual correction is called "corrected" (CORR).

The number of moments by groups for each original experiment is given in Figure 5.1 and the corresponding number of images by groups is given in Figure 5.2. The overall statistics are given in Figure 5.3 and Figure 5.4.

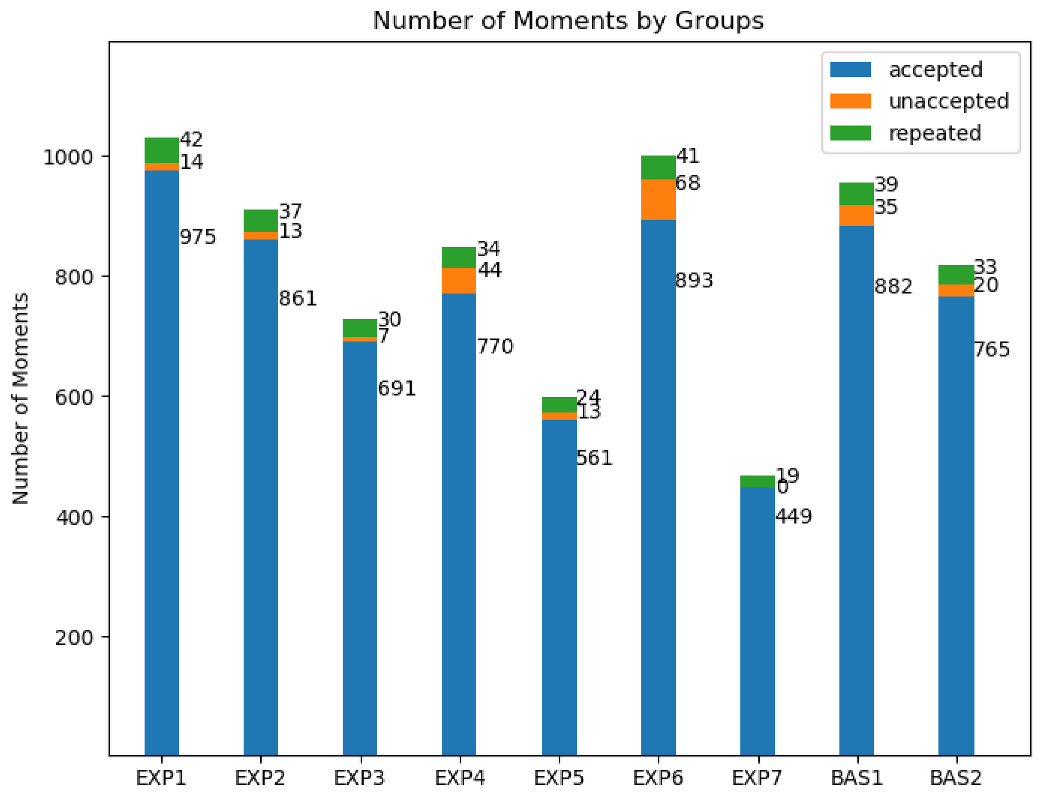


Figure 5.1. Number of Moments by Groups (ORGN)

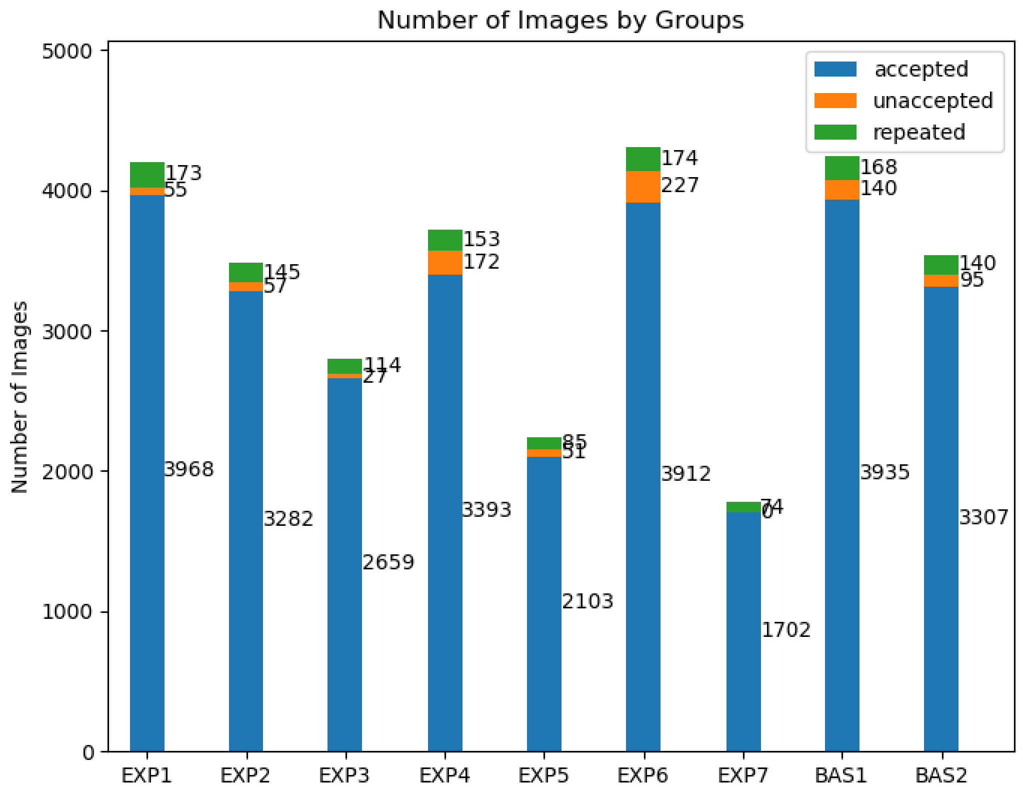


Figure 5.2. Number of Images by Groups (ORGN)

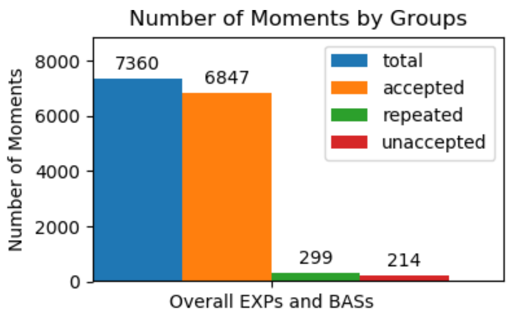


Figure 5.3. Number of Overall Moments by Groups (ORGN)

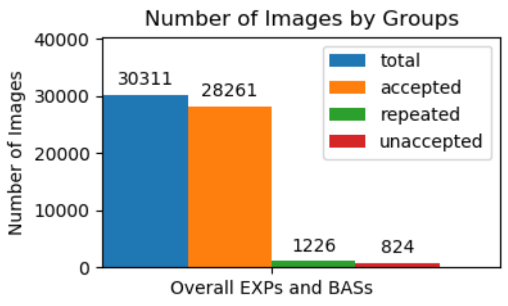


Figure 5.4. Number of Overall Images by Groups (ORGN)

The number of moments by score groups for each original experiment is given in Figure 5.5 and the corresponding number of images by score groups is given in Figure 5.6. The overall statistics of experiments from 1 to 6 are given in Figure 5.7 and Figure 5.8.

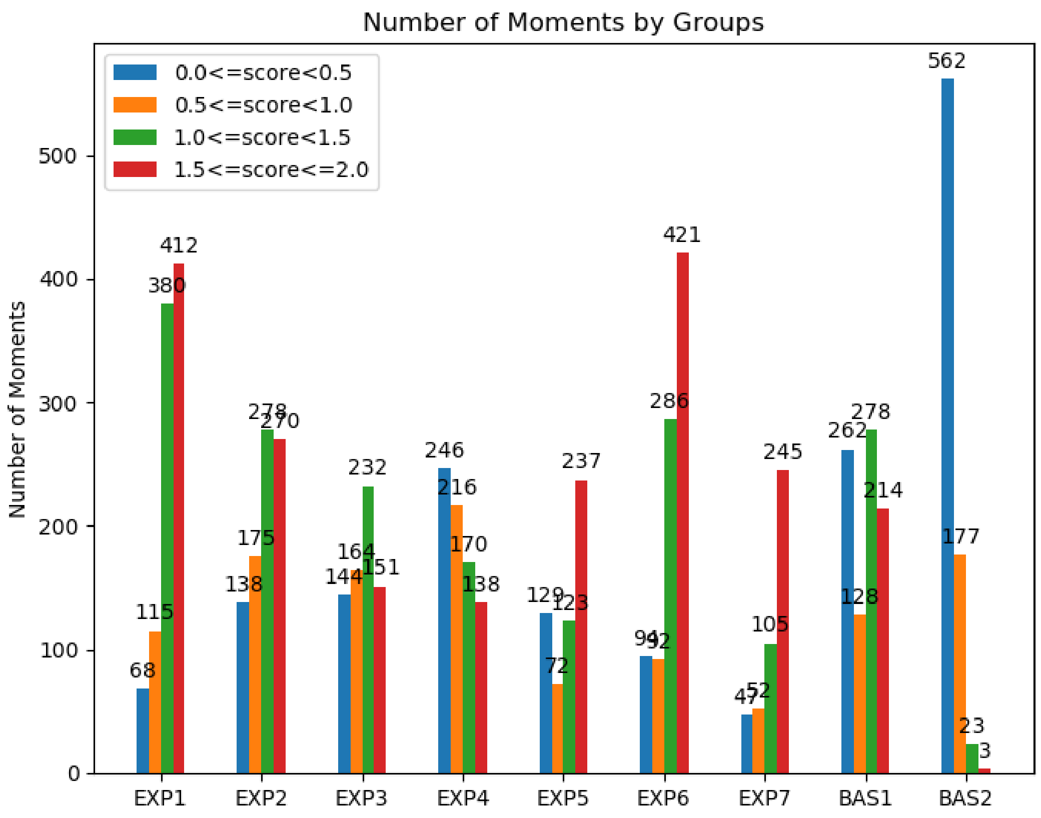


Figure 5.5. Number of Moments by Score Groups (ORGN)

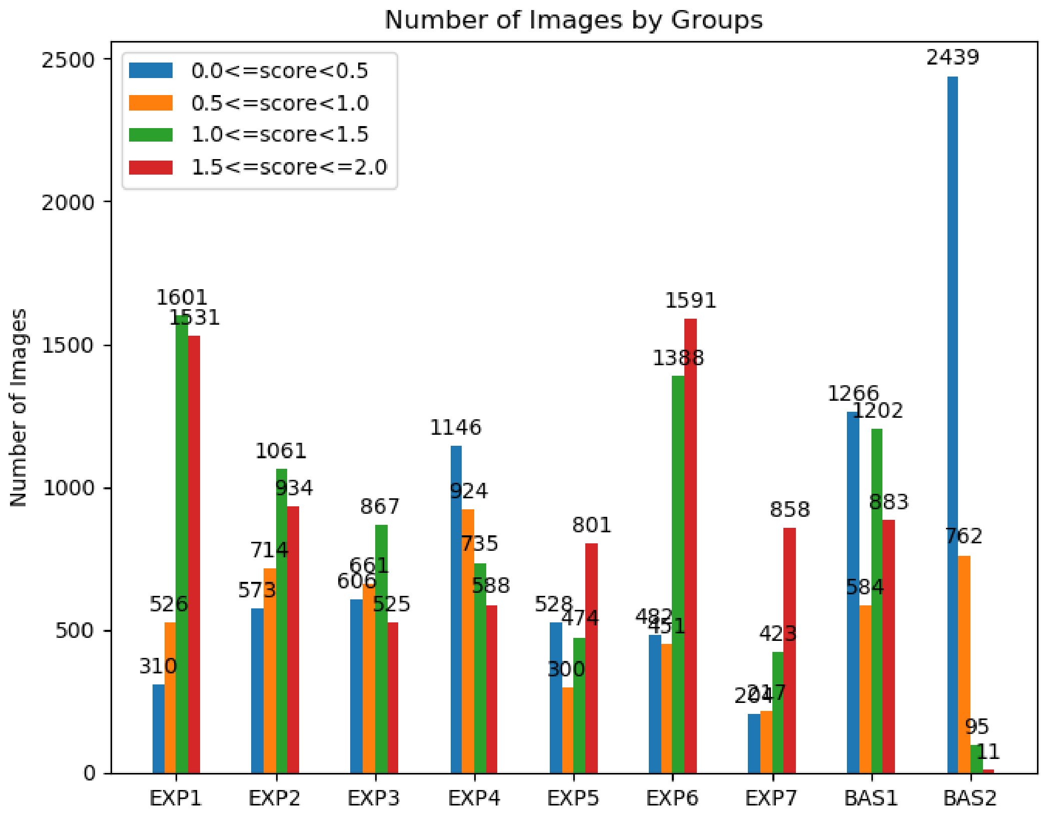


Figure 5.6. Number of Images by Score Groups (ORGN)

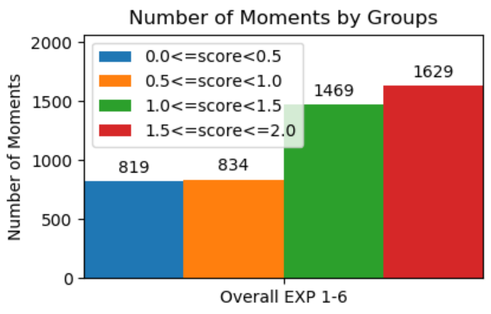


Figure 5.7. Number of Overall EXP 1-6 Moments by Score Groups (ORGN)

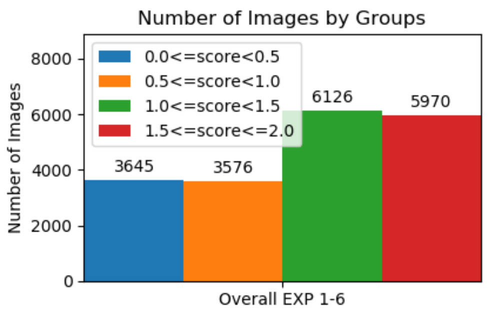


Figure 5.8. Number of Overall EXP 1-6 Images by Score Groups (ORGN)

At this step, it was possible to extract the internal error values of the manual coding process by calculating the differences of each duplicate moments. It is shown in Figure 5.3 that there are 299 repeated moments, however 21 of them were excluded since they were coded as bad or more than two AUs were not coded. Since MAE and RMSE are concerned, it is not essential how the difference (e.g. A-B or B-A) is calculated among duplicate moments' scores. The extracted error distribution of manual coding is given in Figure 5.9. It was calculated from the errors that MAE is equal to 0.169 and RMSE is 0.251.

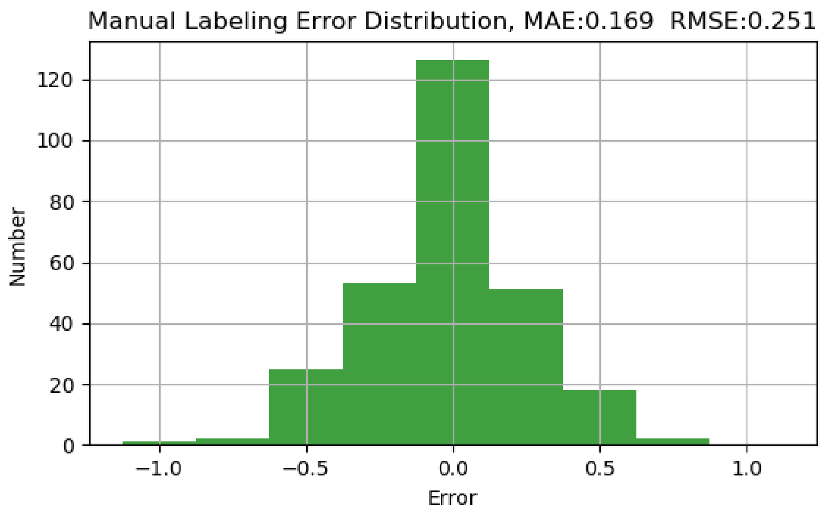


Figure 5.9. Manual Labeling Error Distribution

As the ratio of $\left(\frac{RMSE}{MAE}\right)$ is equal to 1.485 which is higher than the threshold value, 1.375, (described in section 4.8.4) it was decided to use MAE as the loss function and the evaluation criteria for the regression outputs.

5.3. Manual Correction and Image Collection

The moment statistical data given in the previous section, 5.2, is also applicable for the manual correction process since the moment scores were not modified in this step. However, the statistical data of the images may differ from the original.

The number of images by groups after manual correction is given in Figure 5.10. The overall statistics are given in Figure 5.11.

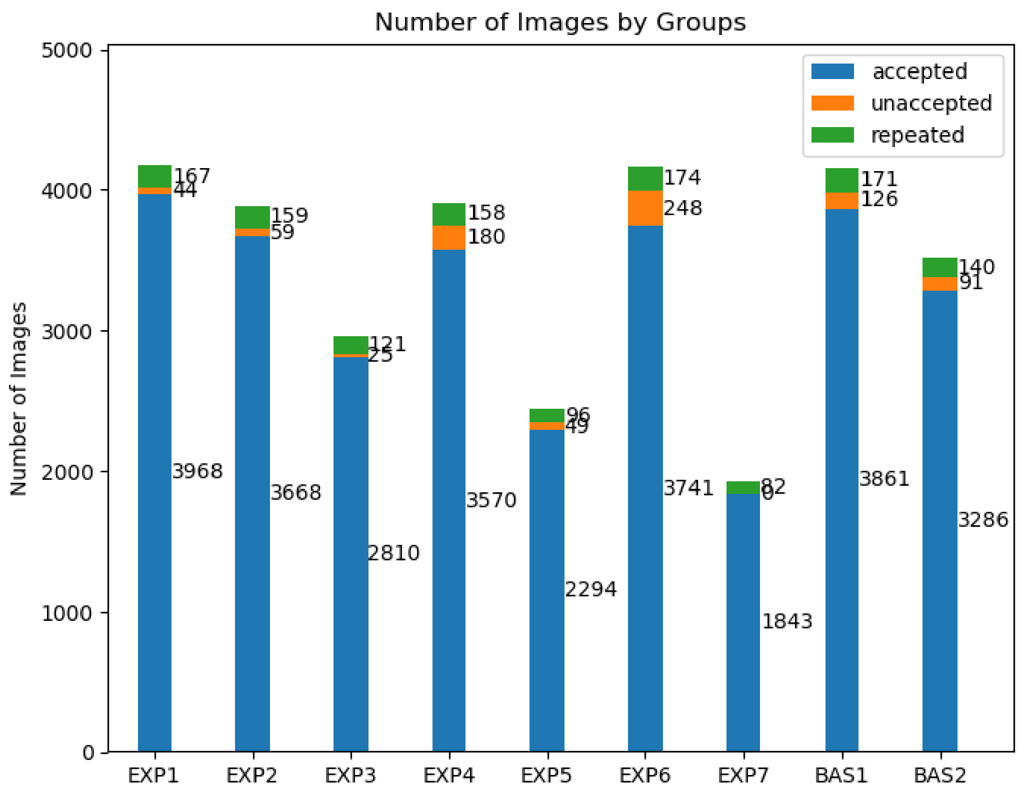


Figure 5.10. Number of Images by Groups (CORR)

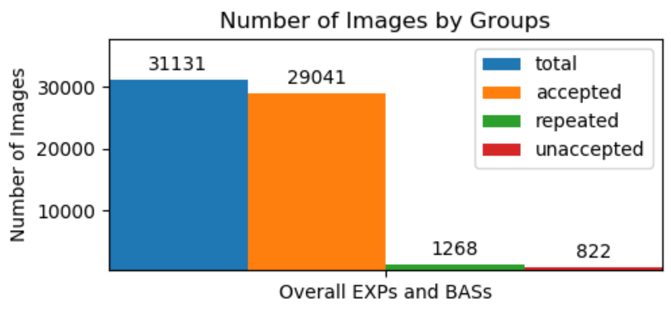


Figure 5.11. Number of Overall Images by Groups (CORR)

The number of images by score groups for each experiment after correction is given in Figure 5.12. The overall statistics of experiments from 1 to 6 are given in Figure 5.13.

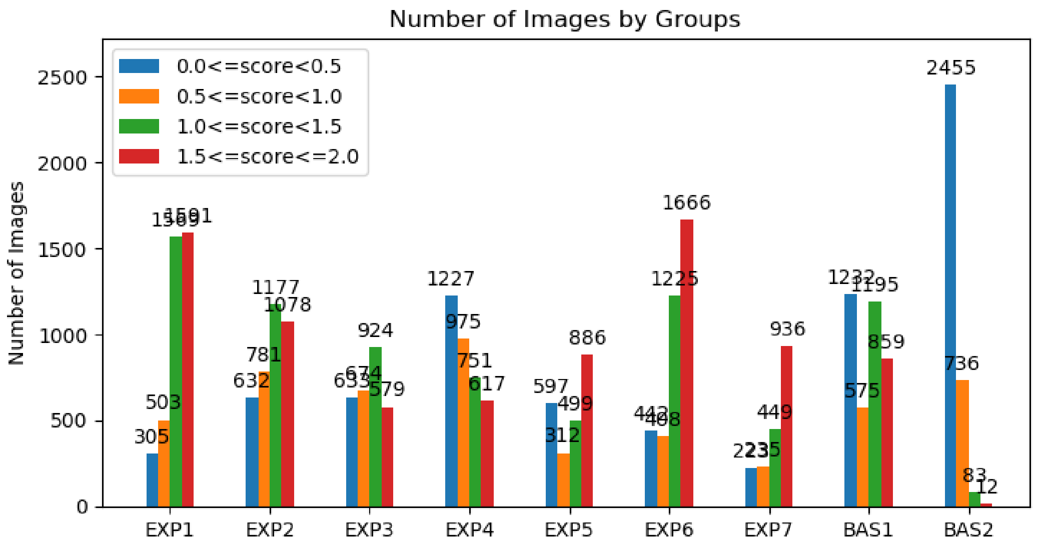


Figure 5.12. Number of Images by Score Groups (CORR)

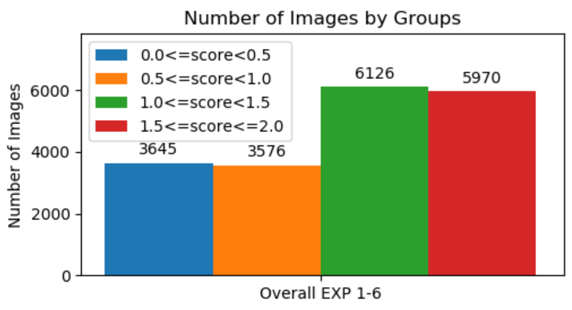


Figure 5.13. Number of Overall EXP 1-6 Images by Score Groups (CORR)

For each of the experiments, “Pain and Basal”, four different folders were able to be created with the corresponding images as described in Section 4.6. However, the proposed cross-validation method as shown in Figure 4.12 requires a total of 29 subfolders belonging to 9 experiment datasets. Since BAS2 was included in only training data, the ORGN folder was not necessary. Similarly, only ORGN folders of EXP7 and BAS1 are needed as they were used as validation data. An extra filtering was applied only for collecting BAS1 images such that 200 moments were selected from the score range 0-0.49, 100 moments were selected from the score range 0.5-0.99 and 50 moments were selected from the score range 1.0-1.49.

The number of images collected for the output folders is summarized in Table 5.4.

Table 5.4. *Number of Images in Output Folders*

Name	*ORGN*	*CORR*	*SYMM*	*AUGM*	***TOTAL***
EXP1	3968	3968	3968	31744	**43648**
EXP2	3282	3668	3668	29344	**39962**
EXP3	2659	2810	2810	22480	**30759**
EXP4	3393	3570	3570	28560	**39093**
EXP5	2103	2294	2294	18352	**25043**
EXP6	3912	3741	3741	29928	**41322**
EXP7	-	1843	-	-	**1843**
BAS1	-	1597	-	-	**1597**
BAS2	-	3286	3286	26288	**32860**
TOTAL	**19317**	**26777**	**23337**	**186696**	**256127**

5.4. Dataset Construction

Six groups of LOSOCV datasets were prepared each of which has four different subsets (“npz” archive files); training, validation, test#1 (test_orgn) and test#2 (test_corr). The content of each LOSOCV datasets is given in Figure 4.12. The size of all the collected images summarized in is around 5.35 GB, however storing those images within “numpy” arrays could take several multiples of that size.

In order to decrease the data processing time during training, the images were resized to 224x224 which is the default size of most of the pre-trained networks in this study. This size was also a reasonable value when compared to the average size of collected images. The file sizes of each archive files used in LOSOCV datasets are summarized in Table 5.5.

Table 5.5. *LOSOCV Dataset File Sizes in MB*

LOSOCV#	*train*	*validation*	*test_orgn*	*test_corr*	***TOTAL***
1	17909	326	375	375	18985
2	18190	326	310	346	19172
3	18991	326	250	265	19832
4	18367	326	311	328	19332
5	19485	326	196	215	20222
6	18118	326	368	353	19165
TOTAL	111060	1956	1810	1882	116708

5.5. Training and Evaluation

5.5.1. First Phase

In the first phase, 5 pre-trained CNNs were compared for both regression and classification outputs. The pre-trained CNNs built up the bottom part of the resulting network, on top of them 2 fully connected layers along with and a final 1-node output layer were added. For classification output, the output layer was changed to 3-node with a “softmax” activation function.

The general structure of the used model for regression output is shown in Figure 5.14.

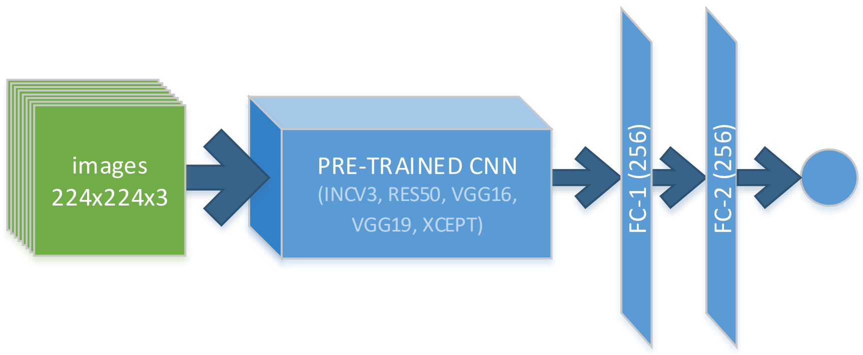


Figure 5.14. First Phase Regression Network General Structure

No pooling was applied after the last layer of the CNN, thus a flattening layer was added between CNN and FC-1. For both FC layers, ReLU is used as the activation function and dropout layers were added just after them. The specification of the used network for regression output is given in Table 5.6.

Table 5.6. *First Phase Regression Network Specification*

Feature	*Value*
Input Size	n x 224 x 224 x 3
Batch Size	32
CNN	Trainable: No
FC-1	Size:256, Activation: ReLU, Dropout=0.15
FC-2	Size:256, Activation: ReLU, Dropout=0.15
Optimizer	Adam (η=1e-4, β_1=0.9, β_2=0.999, decay=0)
Loss Function	MAE
Early Stop Callback	Monitor: Validation Loss, Patience:5
BINS for Image Generators	[0, 0.5, 1.0, 1.5, 2.0]

There are two different train datasets for each LOSOCV iteration which are composed of either ORGN or CORR images. Besides, the image generator, IMGEN1, has a feature for balancing the dataset with the given BINS and IMGEN2 feeds a process without balancing (an image appears only once in an epoch). Thus, four different evaluation values were calculated for each LOSOCV iteration. These were represented as ORGN, ORGN-B, CORR, CORR-B where the suffix, "B", represents the balanced version. The used image generators and their settings for each dataset used in a LOSOCV iteration is given in Table 5.7.

Table 5.7. *Dataset Image Generator Settings*

Dataset	*Image Generator*
Train	IMGEN1 (Shuffle: Yes)
Validation	IMGEN1 (Shuffle: No)
Test (ORGN)	IMGEN2
Test (ORGN-B)	IMGEN1 (Shuffle: No)
Test (CORR)	IMGEN2
Test (CORR-B)	IMGEN1 (Shuffle: No)

The evaluation results of the first phase for regression network are given Table 5.8.

Table 5.8. *First Phase Regression Evaluation Results*

	TEST LOSS (MAE)					
LOSOCV#	**Eval. Type**	**INCV3**	**RES50**	**VGG16**	**VGG19**	**XCEPT**
1	ORGN	0.507	0.774	0.303	0.319	0.470
	ORGN-B	0.443	0.573	0.321	0.316	0.441
	CORR	0.502	0.785	0.276	0.299	0.471
	CORR-B	0.425	0.571	0.300	0.297	0.430
2	ORGN	0.483	0.552	0.324	0.334	0.466
	ORGN-B	0.468	0.520	0.315	0.324	0.466
	CORR	0.479	0.561	0.310	0.306	0.458
	CORR-B	0.459	0.522	0.302	0.296	0.451
3	ORGN	0.362	0.626	0.275	0.291	0.406
	ORGN-B	0.383	0.637	0.280	0.294	0.435
	CORR	0.370	0.639	0.253	0.265	0.405
	CORR-B	0.387	0.640	0.253	0.265	0.427
4	ORGN	0.442	0.494	0.377	0.356	0.446
	ORGN-B	0.476	0.566	0.372	0.358	0.480
	CORR	0.432	0.495	0.343	0.334	0.445
	CORR-B	0.469	0.565	0.340	0.338	0.471
5	ORGN	0.553	0.792	0.351	0.368	0.545
	ORGN-B	0.462	0.643	0.319	0.341	0.479
	CORR	0.559	0.797	0.306	0.325	0.520
	CORR-B	0.461	0.644	0.283	0.305	0.461
6	ORGN	0.728	0.899	0.452	0.432	0.599
	ORGN-B	0.564	0.660	0.404	0.412	0.529
	CORR	0.746	0.938	0.426	0.397	0.601
	CORR-B	0.540	0.661	0.362	0.353	0.486
Mean	ORGN	0.513	0.690	**0.347**	0.350	0.489
	ORGN-B	0.466	0.600	**0.335**	0.341	0.472
	CORR	0.515	0.702	**0.319**	0.321	0.483
	CORR-B	0.457	0.601	**0.307**	0.309	0.454

It is seen in Table 5.8 that the VGGNets, VGG16 and VGG19, performed far better than the other networks for regression type evaluation. The reason for this could be overfitting of the deeper networks.

The specification of the used network for 3-alternative forced classification output is given in Table 5.9.

Table 5.9. *First Phase 3-Alternative Forced Classification Network Specification*

Feature	*Value*
Input Size	n x 224 x 224 x 3
Batch Size	36
CNN	Trainable: No
FC-1	Size:256, Activation: ReLU, Dropout=0.15
FC-2	Size:256, Activation: ReLU, Dropout=0.15
Optimizer	Adam (η=1e-4, β_1=0.9, β_2=0.999, decay=0)
Loss Function	Sparse Categorical Crossentropy
Early Stop Callback	Monitor: Validation Loss, Patience:5
BINS for Image Generators	[0, 0.5, 1.5, 2.0]

As in the regression network, four different evaluation values were calculated for each LOSOCV iteration. The image generators and the settings were used as the same which is given in Table 5.7.

The evaluation results of the first phase for 3-alternative forced network are given in Table 5.10.

Table 5.10. *First Phase 3-Alternative Forced Classification Network Evaluation Results*

	TEST ACCURACY (%)					
LOSOCV#	**Eval. Type**	**INCV3**	**RES50**	**VGG16**	**VGG19**	**XCEPT**
1	ORGN	53.02	7.81	65.05	65.35	40.32
	ORGN-B	40.45	33.33	63.75	61.00	38.70
	CORR	51.74	7.69	67.94	67.31	42.67
	CORR-B	42.31	33.33	66.78	62.19	41.12
2	ORGN	45.58	17.46	58.74	58.32	44.15
	ORGN-B	37.42	33.33	65.86	65.80	38.25
	CORR	45.91	17.23	61.29	60.03	46.43
	CORR-B	38.52	33.33	69.14	67.77	40.51
3	ORGN	57.43	22.79	66.53	64.57	55.43
	ORGN-B	44.27	33.33	69.42	67.99	42.47
	CORR	57.40	22.53	68.26	66.33	55.59
	CORR-B	44.80	33.33	73.01	71.68	43.91
4	ORGN	55.32	33.78	62.95	62.60	53.73
	ORGN-B	41.65	33.33	64.05	63.57	41.37
	CORR	56.72	34.37	65.35	66.97	55.32
	CORR-B	43.71	33.33	66.40	67.15	43.61
5	ORGN	49.12	25.11	74.80	75.65	44.84
	ORGN-B	45.32	33.33	74.50	74.42	41.96
	CORR	48.95	26.02	78.99	77.33	49.08
	CORR-B	46.06	33.33	78.75	76.31	46.40
6	ORGN	43.89	12.32	63.14	65.44	43.69
	ORGN-B	40.37	33.33	60.14	58.30	37.37
	CORR	42.72	11.82	69.07	74.26	49.59
	CORR-B	43.90	33.33	66.85	67.63	45.96
Mean	ORGN	50.73	19.88	65.20	**65.32**	47.03
	ORGN-B	41.58	33.33	**66.29**	65.18	40.02
	CORR	50.57	19.94	68.48	**68.71**	49.78
	CORR-B	43.22	33.33	**70.15**	68.79	43.59

The results presented in Table *5.10* describe that VGGNets performed far better than the other networks for classification type evaluation. This result is in parallel with the regression type evaluation given in Table *5.8*, as expected.

5.5.2. Second Phase

The results in the first phase showed that VGG16 is the winner by far among the other pre-trained networks (except VGG19) for both regression and classification tasks. Although the observations showed a slight difference between VGG16 and VGG19, the first place is not questionable in favor of VGG16. For this reason, VGG16 was used throughout this phase as the pre-trained CNN.

The fine tuning was the major objective of this phase. For the sake of simplicity, the parameter search was carried out for regression type with only one LOSOCV dataset.

The selection among the LOSOCV datasets was not performed in a random fashion. The dataset which has loss values closer to the mean loss of all datasets was chosen. The proximity evaluation was done by calculating the RMSE between the loss and the mean loss value. Since there are four different types of test, the average of the calculated proximities was performed over 4 different test types. Those calculated values are given in Table *5.11*.

Table 5.11. *LOSOCV Dataset Mean Proximity Evaluation*

	TEST LOSS				**ROOT SQUARE OF (LOSS-MEAN LOSS)**				
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**	**MEAN**
1	0.303	0.321	0.276	0.300	0.044	0.014	0.043	0.006	0.027
2	0.324	0.315	0.310	0.302	0.023	0.020	0.009	0.005	0.015
3	0.275	0.280	0.253	0.253	0.072	0.055	0.065	0.054	0.062
4	0.377	0.372	0.343	0.340	0.030	0.037	0.024	0.033	0.031
5	0.351	0.319	0.306	0.283	0.004	0.016	0.013	0.024	**0.014**
6	0.452	0.404	0.426	0.362	0.105	0.068	0.107	0.056	0.084
Mean Loss	0.347	0.335	0.319	0.307					

It was observed that the dataset, LOSOCV-5, is the winner of the six in the competition of closeness to the mean. For this reason, LOSOCV-5 dataset was used for fine tuning network parameters.

5.5.2.1. Normalization

The original paper of VGGNet [29] suggested to use input normalization per channel. This was implemented in this step by taking the average of each channel among all the training images. Then, the difference between image data and mean value was fed to the processes.

Since the images are the inputs of the network in this study, standardization was not used and in fact it is not required since the input values are already restricted.

Adding a batch normalization layer after each fully connected layer was also studied. The default parameters of batch normalization layer of "Keras API" were not modified. Then, the training performance was also compared after implementing both input and batch normalization.

For this step, some parameters of the training process are also modified compared to the first phase. The modified parameters are given in Table *5.12*.

Table 5.12. *Modified Network Parameters*

Feature	*Value*
FC-1	Size:256, Activation: ReLU, Dropout=0.20
FC-2	Size:256, Activation: ReLU, Dropout=0.20
Early Stop Callback	Monitor: Validation Loss, Patience:8

The result comparison between "no normalization" and several normalization combinations for LOSOCV-5 dataset is given in Table *5.13*.

Table 5.13. *Different Normalization Performance Comparison of LOSOCV-5*

TEST LOSS (MAE)				
Eval. Type	**NN**	**IN**	**BN**	**IN + BN**
ORGN	0.351	0.388	0.330	**0.324**
ORGN-B	0.319	0.343	0.330	**0.309**
CORR	0.306	0.359	0.294	**0.279**
CORR-B	0.283	0.318	0.298	**0.273**

NN: No Normalization, IN: Input Normalization, BN: Batch Normalization

It was observed that implementing both input and batch normalization increased the performance of the network. Applying a single normalization did not perform better. This makes sense in such a way that normalization takes effect when implemented in all applicable layers. Thus, it was decided to use both normalizations for the next step.

The resulting network structure is illustrated in Figure 5.15.

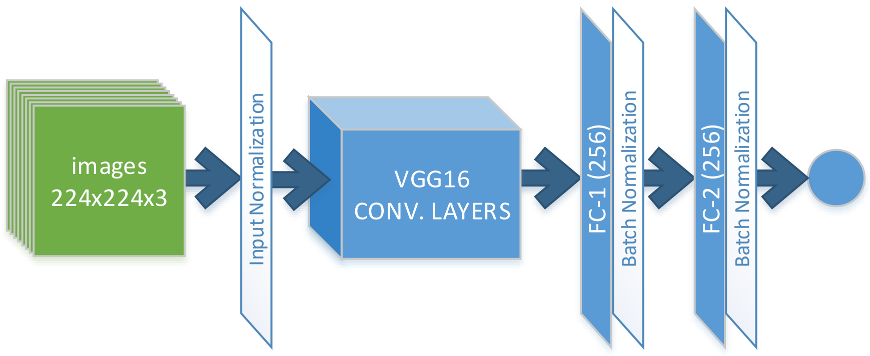


Figure 5.15. Network Structure with Normalization

5.5.2.2. Fully Connected Layers

Up to this point, the general structure of fully connected layers was not modified such that two FC layers were used with number of nodes, 256. In this step, the performances of some other combinations of FC layers were studied.

The results of 9 different configurations of FC layers are given in Table *5.14*.

Table 5.14. *Fully Connected Layer Configuration Performances*

FC Layer Configuration		**LOSOCV-5 TEST LOSS (MAE)**			
# Layers	**Layer Size**	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	128	0.343	0.320	0.300	0.287
1	256	0.325	0.313	0.292	0.285
1	512	0.320	0.317	0.284	0.289
2	128	0.353	0.327	0.308	0.291
2	256	0.324	**0.309**	**0.279**	**0.273**
2	512	0.321	0.312	0.281	0.279
3	128	0.391	0.351	0.351	0.319
3	256	0.314	0.311	0.273	0.279
3	512	**0.312**	0.315	0.268	0.281

Although small differences were observed on the performances of different fully connected layer structures, the configuration of 2-layer with 256 nodes structure was still the best one. For this reason, this configuration was chosen for the next step.

5.5.2.3. Learning Rate and Activation Function

Up to this step, ReLU had been used for the activation functions of fully connected layers, and a fixed value of 1e-4 had been used for the learning rate of Adam optimizer.

In this step, some other combinations of activations functions and learning rate parameters were studied. For learning rate, 4 different values (1e-2, 1e-3, 1e-4, 1e-5 and 1e-6) and for activation function, 3 different Leaky ReLU functions with alpha (slope of negative range) values of 0, 0.1 and 0.2 were used. It is better to state that a Leaky ReLU with alpha equals to zero corresponds to classical ReLU function.

The results of 15 different combinations of learning rate and activation functions are given in Table *5.15*.

Table 5.15. *Learning Rate and Activation Function Combination Performances*

PARAMETERS		**LOSOCV-5 TEST LOSS (MAE)**			
Adam Optimizer Learning Rate	**Leaky ReLU Alpha**	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1.E-02	0.0	0.331	0.324	0.294	0.294
1.E-02	0.1	0.317	0.298	0.292	0.280
1.E-02	0.2	0.311	0.300	0.280	0.276
1.E-03	0.0	0.359	0.328	0.316	0.296
1.E-03	0.1	0.293	**0.292**	**0.258**	**0.265**
1.E-03	0.2	0.304	0.295	0.268	0.266
1.E-04	0.0	0.324	0.309	0.279	0.273
1.E-04	0.1	**0.291**	0.297	0.262	0.273
1.E-04	0.2	0.323	0.301	0.290	0.274
1.E-05	0.0	0.360	0.331	0.324	0.304
1.E-05	0.1	0.315	0.296	0.278	0.266
1.E-05	0.2	0.355	0.330	0.305	0.290
1.E-06	0.0	0.359	0.334	0.316	0.298
1.E-06	0.1	0.350	0.325	0.303	0.288
1.E-06	0.2	0.361	0.328	0.309	0.289

It was observed that the best result was achieved with the learning rate of 1e-3 and Leaky ReLU alpha parameter of 0.1. This configuration was also compared with 2 different configurations in which learning decay parameter was set to 5e-5 value instead of using default 0.0 value.

The results of inserting learning rate decay and performance comparison of 3 different configurations are given in Table *5.16*.

Table 5.16. *Learning Rate Decay Parameter Comparison*

PARAMETERS			**LOSOCV-5 TEST LOSS (MAE)**			
Adam Optimizer Learning Rate	**Leaky ReLU Alpha**	**Learning Rate Decay**	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1.E-03	0.1	0.0E+00	**0.293**	**0.292**	**0.258**	**0.265**
1.E-03	0.1	5.0E-05	0.314	0.307	0.283	0.281
1.E-02	0.1	5.0E-05	0.305	0.292	0.272	0.269

It was observed that inserting learning rate decay did not change the best result. Thus, 1e-3 was used for learning rate without additional decay parameter for the next step and also Leaky ReLU with an alpha value of 0.1 was used as an activation function.

5.5.2.4. Regression Network Results

The summary of the resulting regression network, designated as $N_R^{(1)}$, after studying different network parameters is given in Table *5.17*.

Table 5.17. $N_R^{(1)}$, *Network Summary*

Feature	*Value*
Input Size	n x 224 x 224 x 3
Batch Size	32
CNN	Trainable: No
FC-1	Size:256, Activation: Leaky ReLU(α=0.1) Dropout=0.2
FC-2	Size:256, Activation: Leaky ReLU(α=0.1) Dropout=0.2
Optimizer	Adam (η=1e-3, β_1=0.9, β_2=0.999, decay=0)
Loss Function	MAE
Early Stop Callback	Monitor: Validation Loss, Patience:8
BINS for Image Generators	[0, 0.5, 1.0, 1.5, 2.0]
Normalization	Input Normalization, Batch Normalization

The other LOSOCV datasets were also trained with the network $N_R^{(1)}$ in order to get the mean cross-validation result. These results are given in Table *5.18*.

Table 5.18. $N_R^{(1)}$, *Cross Validation Results*

	TEST LOSS (MAE)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	0.304	0.343	0.284	0.317
2	0.300	0.297	0.284	0.280
3	0.307	0.307	0.294	0.289
4	0.426	0.406	0.386	0.366
5	0.293	0.292	0.258	0.265
6	0.376	0.396	0.324	0.324
MEAN	0.334	0.340	0.305	0.307

It was observed that even though the results of LOSOCV-5 had shown an improvement, overall results did not show any improvement. This is an indication that the fine-tuning of the fully-connected layers has limited impact and is highly specific to the selected test dataset.

The BINS of the image generator of the network $N_R^{(1)}$ were changed to 8 divisions ([0, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0]) and the resulting network is called $N_R^{(2)}$. The cross-validation results of $N_R^{(2)}$ are given in Table 5.19.

Table 5.19. $N_R^{(2)}$, *Cross Validation Results*

	TEST LOSS (MAE)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	0.288	0.353	0.270	0.327
2	0.313	0.325	0.297	0.306
3	0.306	0.307	0.291	0.289
4	0.363	0.344	0.330	0.315
5	0.319	0.293	0.284	0.270
6	0.396	0.385	0.352	0.333
MEAN	0.331	0.335	0.304	0.307

No remarkable improvement was observed by increasing the divisions of the BIN list of the image generator. This shows that four divisions used for balancing the mini-batches are sufficient for our dataset.

The convolutional layers of the networks $N_R^{(1)}$ and $N_R^{(2)}$ are not trainable. A different network configuration, designated as $N_R^{(3)}$, was also studied with changing the property of the convolutional layers as trainable. The summary of $N_R^{(3)}$ is given in Table 5.20.

Table 5.20. $N_R^{(3)}$, *Network Summary*

Feature	*Value*
Input Size	n x 224 x 224 x 3
Batch Size	32
CNN	Trainable: Yes
FC-1	Size:256, Activation: Leaky ReLU(α=0.1) Dropout=0.2
FC-2	Size:256, Activation: Leaky ReLU(α=0.1) Dropout=0.2
Optimizer	Adam (η=1e-5, β_1=0.9, β_2=0.999, decay=0)
Loss Function	MAE
Early Stop Callback	Monitor: Validation Loss, Patience:8
BINS for Image Generators	[0, 0.5, 1.0, 1.5, 2.0]
Normalization	Input Normalization, Batch Normalization

The cross-validation results of $N_R^{(3)}$ are given in Table *5.21*.

Table 5.21. $N_R^{(3)}$, *Cross-Validation Results*

	TEST LOSS (MAE)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	0.272	0.314	0.256	0.294
2	0.274	0.276	0.264	0.266
3	0.272	0.271	0.265	0.259
4	0.309	0.316	0.277	0.286
5	0.296	0.276	0.269	0.253
6	0.331	0.355	0.295	0.310
MEAN	0.293	0.301	0.271	0.278

A performance increase was achieved by setting convolutional layers as trainable. This is an expected result since the dataset used in this study is dissimilar to the ImageNet dataset which was used to train the VGG16 network.

5.5.2.5. Regression Network Multi-Camera Evaluation

For multi-camera evaluation, the average value of the prediction values belonging to the same moment was calculated. Then, the MAE of each LOSOCV dataset was calculated. The equal-weighted (balanced) MAE (designated with suffix, "B") was also calculated by grouping the scores first, calculating the MAE within the groups and then taking the average of the MAEs of the groups.

The multi-camera cross-validation results of $N_R^{(1)}$ are summarized in Table *5.22*.

Table 5.22. $N_R^{(1)}$, *Multi-Camera Cross-Validation Results*

	TEST LOSS (MAE)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	0.252	0.322	0.240	0.295
2	0.255	0.268	0.241	0.254
3	0.268	0.271	0.264	0.267
4	0.404	0.361	0.369	0.332
5	0.253	0.244	0.222	0.226
6	0.316	0.354	0.279	0.301
MEAN	0.291	0.303	0.269	0.279

The multi-camera cross-validation results of $N_R^{(3)}$ are summarized in Table *5.23*.

Table 5.23. $N_R^{(3)}$, *Multi-Camera Cross-Validation Results*

	TEST LOSS (MAE)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	0.229	0.296	0.220	0.272
2	0.237	0.254	0.235	0.250
3	0.247	0.250	0.245	0.245
4	0.277	0.269	0.254	0.251
5	0.285	0.252	0.260	0.235
6	0.297	0.337	0.278	0.307
MEAN	0.262	0.276	0.249	0.260

The multi-camera cross-validation results of $N_R^{(2)}$ are not given, since no considerable performance change was observed compared to the network, $N_R^{(1)}$.

As expected, performance increase was achieved using multi-camera method for regression type evaluation. The reason is that taking average of the predictions of multiple frames leads to a decrease in error.

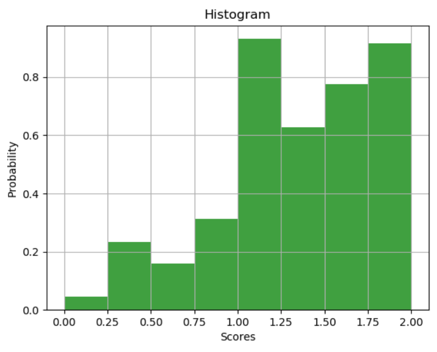
The correlation coefficient, “Pearson’s r”, analysis was carried out for the network, $N_R^{(3)}$, using the multi-camera method predictions. The results of this study are presented in Table 5.24.

Table 5.24. $N_R^{(3)}$, *Multi-Camera Correlation Analysis*

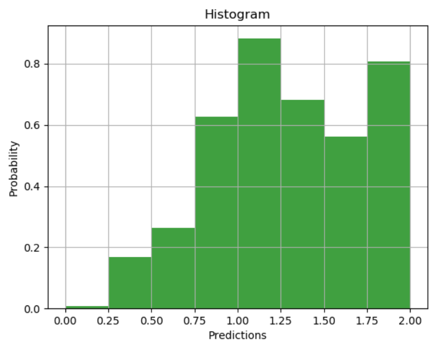
LOSOCV#	**1**	**2**	**3**	**4**	**5**	**6**	**Overall**
Pearson's r	0.80	0.86	0.86	0.87	0.92	0.86	**0.86**

A complete set of statistics belonging to TEST-CORR dataset for multi-camera evaluation of the network, $N_R^{(3)}$, is given in Appendix A. For clarification, the prediction related statistics of LOSOCV-1 TEST-CORR is given as an example in Figure 5.16. The distribution of both manual scores and the predictions are given in

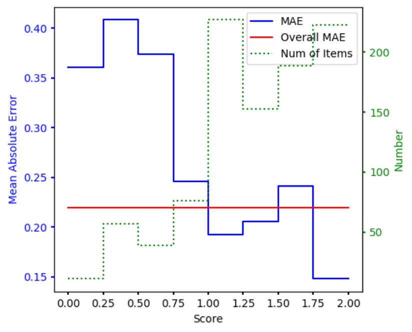
Figure 5.16(a) and Figure 5.16(b). The calculated MAE and the number of moments within different score groups along with the overall MAE are given in Figure 5.16(c). The calculated ME and the number of moments within different score groups along with the overall ME are given in Figure 5.16(d). In Figure 5.16(e) and Figure 5.16(f), the error distribution is given in two different ways. In the former one, a probabilistic histogram is used with 20 divisions. However, in the latter one, a stacked histogram with 12 divisions is used to give more detail for the error based on score groups.



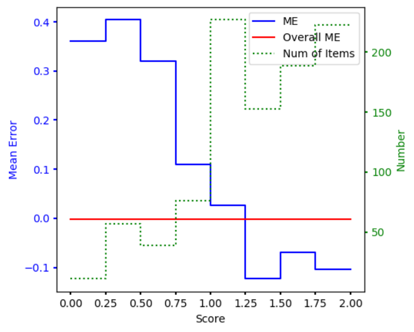
(a)



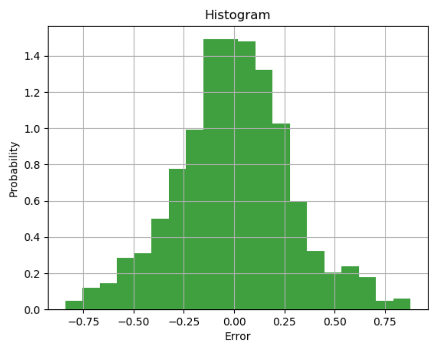
(b)



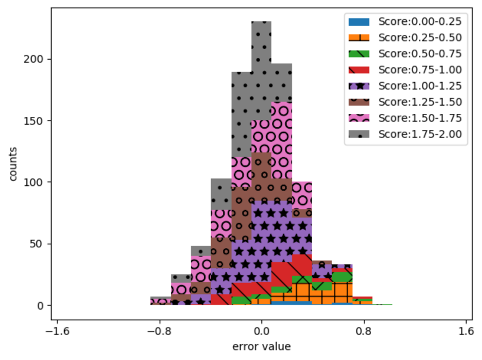
(c)



(d)



(e)



(f)

Figure 5.16. $N_R^{(3)}$ LOSOCV-1 TEST-CORR Multi-Camera Statistics; (a) Moment Score Distribution, (b) Moment Prediction Distribution, (c) MAE vs Score Groups and Number of Moments Distribution, (d) ME vs Score Groups and Number of Moments Distribution, (e) Error Distribution, (f) Error Distribution with Score Groups

5.5.2.6. Regression Output to Binary Classification Results

Since the result of the regression network is a scalar value, it may be desired to analyze the classification performance of the regression network by mapping the scalar to the classification score ranges. For this purpose, the multi-camera results of two regression networks, $N_R^{(1)}$ and $N_R^{(3)}$, were mapped to the binary (pain/no-pain) classification ranges (scores lower than 0.5 were designated as no-pain and scores higher than or equal to 0.5 were designated as pain).

The multi-camera cross-validation results of $N_R^{(1)}$ after regression to binary conversion are summarized in Table 5.25.

Table 5.25. $N_R^{(1)}$, *Regression to Binary Classification, Multi-Camera Cross-Validation Results*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	92.00	55.57	92.10	61.07
2	84.44	66.69	85.71	69.51
3	82.92	62.35	85.82	70.83
4	70.78	54.59	73.51	59.51
5	86.45	73.81	90.91	84.85
6	90.26	54.66	91.49	64.27
MEAN	84.48	61.28	86.59	68.34

The multi-camera cross-validation results of $N_R^{(3)}$ after regression to binary conversion are summarized in Table 5.26.

Table 5.26. $N_R^{(3)}$, *Regression to Binary Classification, Multi-Camera Cross-Validation Results*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	92.615	58.622	92.718	63.438
2	85.830	70.455	86.063	73.232
3	86.107	71.527	87.844	76.973
4	80.519	71.885	82.468	76.335
5	86.096	72.214	88.770	78.844
6	89.922	52.597	90.370	55.663
MEAN	86.848	66.217	88.039	70.747

For all networks, discretizing regression to binary output presented very poor performance considering the balanced test cases. The reason is that the slight output value change of regression network may easily cause a wrong prediction for classification evaluation after discretizing. The higher results of the other two test cases were not taken into consideration here, since the accuracy differences between them and their corresponding balanced test cases are quite much.

5.5.2.7. Binary Classification Network Results

Different score mappings were studied for binary (pain/no-pain) classification. In other words, different separations from the linear pain scale were used in order to match scores to binary designation.

For the network, designated as $N_{BC}^{(1)}$, no gap is used for binary-score mapping. In other words, all the images are used while training without any filtering based on scores.

The summary of the network, $N_{BC}^{(1)}$, is given in Table *5.27*.

Table 5.27. $N_{BC}^{(1)}$, *Network Summary*

Feature	*Value*
Input Size	n x 224 x 224 x 3
Batch Size	32
CNN	Trainable: No
FC-1	Size:128, Activation: Leaky ReLU(α=0.1) Dropout=0.5
FC-2	Size:18, Activation: Leaky ReLU(α=0.1) Dropout=0.5
Optimizer	Adam (η=1e-4, β_1=0.9, β_2=0.999, decay=0)
Loss Function	Binary Cross-entropy
Early Stop Callback	Monitor: Validation Accuracy, Patience:8
Binary-Score Mapping	No-Pain(0): score<0.5 Pain(1): score≥0.5
Normalization	-

The binary classification performance summary of the network $N_{BC}^{(1)}$ for all cross-validation tests is given in Table *5.28*.

Table 5.28. $N_{BC}^{(1)}$, *Cross Validation Results*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	80.90	73.57	83.72	76.61
2	72.49	76.43	73.17	78.57
3	79.54	78.46	80.46	80.79
4	75.83	73.52	77.11	76.23
5	82.79	83.02	84.09	85.92
6	79.98	69.61	82.52	75.47
MEAN	78.59	75.77	80.18	78.93

The images having the scores in the range, [0.5, 1.0), were discarded in training for a different network configuration designated as $N_{BC}^{(2)}$. The summary of the network, $N_{BC}^{(2)}$, is given in Table *5.29*.

Table 5.29. $N_{BC}^{(2)}$, *Network Summary*

Feature	*Value*
Input Size	n x 224 x 224 x 3
Batch Size	32
CNN	Trainable: No
FC-1	Size:128, Activation: Leaky ReLU(α=0.1) Dropout=0.5
FC-2	Size:18, Activation: Leaky ReLU(α=0.1) Dropout=0.5
Optimizer	Adam (η=1e-4, β_1=0.9, β_2=0.999, decay=0)
Loss Function	Binary Cross-entropy
Early Stop Callback	Monitor: Validation Accuracy, Patience:8
Binary-Score Mapping	No-Pain(0): score<0.5 Pain(1): score≥1.0
Normalization	-

The binary classification performance summary of the network $N_{BC}^{(2)}$ for all cross-validation tests is given in Table *5.30*.

Table 5.30. $N_{BC}^{(2)}$, *Cross Validation Results*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	81.43	74.28	83.34	74.75
2	68.46	77.80	68.92	79.16
3	79.58	79.27	80.21	80.84
4	76.04	72.63	77.06	74.80
5	77.98	82.78	80.25	85.35
6	73.65	71.78	77.36	80.01
MEAN	76.19	76.42	77.86	79.15

The network, $N_{BC}^{(2)}$, was also tested with discarding the images having the scores in the range, [0.5, 1.0), as in its training process. The binary classification performance summary of the network $N_{BC}^{(2)}$ for all cross-validation tests with discarding those images is given in Table *5.31*.

Table 5.31. $N_{BC}^{(2)}$, *Cross Validation Results, Discarded Score Range [0.5,1.0)*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	85.53	76.65	87.30	77.04
2	80.26	84.49	80.95	85.93
3	87.29	84.87	88.44	86.57
4	78.17	77.10	81.16	80.46
5	85.52	87.55	88.09	90.34
6	76.91	73.72	81.34	82.25
MEAN	82.28	80.73	84.55	83.77

The ignored score region was widened for a different network configuration designated as $N_{BC}^{(3)}$, the images having the scores in the medium score range [0.5, 1.5) were discarded in training process.

The summary of the network, $N_{BC}^{(3)}$, is given in Table *5.32*.

Table 5.32. $N_{BC}^{(3)}$*, Network Summary*

Feature	*Value*
Input Size	n x 224 x 224 x 3
Batch Size	32
CNN	Trainable: No
FC-1	Size:128, Activation: Leaky ReLU(α=0.1) Dropout=0.5
FC-2	Size:18, Activation: Leaky ReLU(α=0.1) Dropout=0.5
Optimizer	Adam (η=1e-4, β_1=0.9, β_2=0.999, decay=0)
Loss Function	Binary Cross-entropy
Early Stop Callback	Monitor: Validation Accuracy, Patience:8
Binary-Score Mapping	No-Pain(0): score<0.5 Pain(1): score≥1.5
Normalization	-

The binary classification performance summary of the network $N_{BC}^{(3)}$ for all cross-validation tests is given in Table *5.33*.

Table 5.33. $N_{BC}^{(3)}$, *Cross Validation Results*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	67.04	75.18	70.46	78.74
2	60.94	73.52	62.21	75.92
3	66.57	75.79	69.07	78.53
4	66.25	72.30	67.28	73.85
5	72.56	79.73	74.54	82.14
6	59.76	70.10	62.82	76.86
MEAN	65.52	74.44	67.73	77.67

The network, $N_{BC}^{(3)}$, was also tested with discarding the images having the scores in the range, [0.5, 1.5), as in its training process. The binary classification performance summary of the network $N_{BC}^{(3)}$ for all cross-validation tests with discarding those images is given in Table *5.34*.

Table 5.34. $N_{BC}^{(3)}$, *Cross Validation Results, Discarded Score Range [0.5, 1.5)*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	87.34	86.34	90.77	89.86
2	89.12	89.84	91.17	92.34
3	93.63	93.70	96.20	96.23
4	90.89	90.87	94.90	94.92
5	94.28	94.26	96.97	97.13
6	85.77	85.09	90.89	92.49
MEAN	90.17	90.02	93.48	93.83

It is seen that the classification performance increases if the images within the medium score range are discarded for test dataset. This indicates that the network could perform better if the ambiguous images are removed.

5.5.2.8. Binary Classification Network Multi-Camera Evaluation

For multi-camera evaluation, the average value of the prediction values (probabilities for pain/no-pain) belonging to the same moment was calculated. Then, the overall accuracy of each LOSOCV dataset was calculated. The equal-weighted (balanced) accuracy (designated with suffix, “B”) was also calculated by first grouping the scores, calculating the accuracy within the groups and then taking the average of the accuracy values of the groups.

The multi-camera cross-validation results of $N_{BC}^{(1)}$, $N_{BC}^{(2)}$, and $N_{BC}^{(3)}$ are summarized in Table 5.35, Table 5.36, and Table 5.37, respectively.

Table 5.35. $N_{BC}^{(1)}$, *Multi-Camera Cross-Validation Results*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	87.20	75.40	87.40	75.50
2	79.20	83.50	78.20	83.80
3	86.40	85.30	85.40	84.60
4	80.40	75.80	79.90	77.30
5	90.00	91.10	88.80	91.10
6	88.40	70.00	88.90	79.70
MEAN	85.27	80.18	84.77	82.00

Table 5.36. $N_{BC}^{(2)}$, *Multi-Camera Cross-Validation Results*

LOSOCV#	TEST ACCURACY (%) ORGN	ORGN-B	CORR	CORR-B
1	87.10	80.10	87.50	77.60
2	72.80	83.20	71.80	82.60
3	85.40	85.10	84.40	84.80
4	81.30	75.70	80.10	76.70
5	83.40	88.70	83.20	88.80
6	82.20	77.80	82.60	84.20
MEAN	82.03	81.77	81.60	82.45

Table 5.37. $N_{BC}^{(3)}$, *Multi-Camera Cross-Validation Results*

LOSOCV#	TEST ACCURACY (%) ORGN	ORGN-B	CORR	CORR-B
1	72.10	83.00	74.40	82.10
2	62.10	77.20	63.10	77.70
3	68.90	79.60	70.00	80.10
4	67.30	75.50	67.40	75.70
5	75.60	83.90	75.60	83.90
6	64.20	78.60	62.60	77.70
MEAN	68.37	79.63	68.85	79.53

As expected, performance increase was achieved using multi-camera method for classification type evaluation (similar to regression case). The reason is that taking average of the predictions of multiple frames leads to an overall increase in prediction accuracy.

The network, $N_{BC}^{(2)}$ is also tested with discarding the moments having the scores in the range, [0.5, 1.0), as in its training process. Similarly, the network, $N_{BC}^{(3)}$, is tested with discarding the moments having the scores in the range [0.5, 1.5). The binary classification multi-camera performance summary of the networks, $N_{BC}^{(2)}$ and $N_{BC}^{(3)}$, for all cross-validation tests with discarding those moments are given in Table *5.38* and Table *5.39*, respectively.

Table 5.38. $N_{BC}^{(2)}$, *Multi-Camera Cross-Validation Results, Discarded Scores [0.5, 1.0)*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	90.60	82.10	90.90	79.60
2	85.60	90.40	84.50	89.80
3	92.80	90.30	92.40	90.20
4	81.40	79.30	83.40	81.80
5	91.20	93.50	91.40	93.90
6	84.60	79.30	86.60	86.40
MEAN	87.70	85.82	88.20	86.95

Table 5.39. $N_{BC}^{(3)}$, *Multi-Camera Cross-Validation Results, Discarded Scores [0.5, 1.5)*

	TEST ACCURACY (%)			
LOSOCV#	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
1	93.10	94.20	94.20	92.90
2	92.90	94.50	93.60	95.00
3	98.30	98.30	98.00	97.90
4	98.20	98.10	98.70	98.70
5	97.50	97.90	98.60	98.80
6	94.40	95.30	93.80	95.00
MEAN	95.73	96.38	96.15	96.38

5.5.2.9. Overall Regression Networks Comparison

The overall regression output performance comparison between the developed network models is given in Table *5.40*.

Table 5.40. *Overall Regression Networks Comparison*

		TEST MAE (%)			
Network	**Multi-Camera**	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
$N_R^{(3)}$	Yes	**0.262**	**0.276**	**0.249**	**0.260**
$N_R^{(3)}$	No	0.293	0.301	0.271	0.278
$N_R^{(1)}$	Yes	0.291	0.303	0.269	0.279
$N_R^{(1)}$	No	0.334	0.34	0.305	0.307
$N_R^{(2)}$	No	0.331	0.335	0.304	0.307

As seen in Table *5.40*, the network, $N_R^{(3)}$, performed better than the others. The most important difference of this network is that the convolutional layers were set as trainable. Additionally, it is observed that multi-camera method is useful to increase the performance of the network one step further.

5.5.2.10. Overall Binary Classification Comparison

The overall binary classification performance comparison between the developed network models is given in Table *5.41*.

Table 5.41. *Overall Binary Classification Comparison*

			TEST ACCURACY (%)			
Network	**Multi-Camera**	**Discarded Scores**	**ORGN**	**ORGN-B**	**CORR**	**CORR-B**
$N_{BC}^{(3)}$	Yes	[0.5, 1.5)	**95.73**	**96.38**	**96.15**	**96.38**
$N_{BC}^{(3)}$	No	[0.5, 1.5)	90.17	90.02	93.48	93.83
$N_{BC}^{(2)}$	Yes	[0.5, 1.0)	87.70	85.82	88.20	86.95
$N_{BC}^{(2)}$	No	[0.5, 1.0)	82.28	80.73	84.55	83.77
$N_{BC}^{(2)}$	Yes	-	82.03	**81.77**	81.60	**82.45**
$N_{BC}^{(1)}$	Yes	-	85.27	80.18	84.77	82.00
$N_{BC}^{(3)}$	Yes	-	68.37	79.63	68.85	79.53
$N_{BC}^{(2)}$	No	-	76.19	76.42	77.86	79.15
$N_{BC}^{(1)}$	No	-	78.59	75.77	80.18	78.93
$N_{BC}^{(3)}$	No	-	65.52	74.44	67.73	77.67
$N_{R}^{(3)}$	Yes	-	**86.85**	66.22	**88.04**	70.78
$N_{R}^{(1)}$	Yes	-	84.48	61.28	86.59	68.34

It is seen from the Table *5.41* that discarding the medium score range from the test dataset provides a great increase in the binary classification accuracy. However, for most practical implementations, this evaluation type (discarding some images from test dataset) may not be desired or may not be a valid method. For this reason, more emphasis was given on the other evaluation results within this study. Among those, the network, $N_{BC}^{(2)}$, achieved the best accuracy, 82.45%, using the multi-camera method for one of the balanced test cases. This result also proved that using multi-camera method could provide a considerable increase in the classification performance of the network.

CHAPTER 6

CONCLUSIONS

The automation of the MGS does not have a long history, since MGS was first introduced in 2012 [1]. For this reason, there have been only a few studies in this area. This study is one of those studies starting with a motivation that the power of convolutional neural networks could perfectly fit the automation task of MGS.

As similar to other image recognition tasks, the most important part of this study is to collect the dataset required for training. Most of the workload was spent on the dataset construction in order to come up with a reliable and feasible solution. The tasks were separated such that they could be considered as building blocks of a complete solution.

First, the procedure and the physical environment of the mouse experiment were described including how the video recordings were taken. Then, the video processing task was explained in detail. The algorithms and methods for the automation of the dataset construction for manual coding were given. After that, manual coding process was introduced. Having the coded moments, image generation task including the data augmentation function was described. The method for validating the results and constructing the training datasets according to this method were stated just before describing proposed training and evaluation plan.

In the experimental results, first, the specifications of the 9 video files of 7 different experiments used in this study were given. Then, the general statistics of the constructed datasets for manual scoring process were included. After manual scoring, the statistical data dependent on the scores were also given for each experiment.

Since each moment was scored by only one expert, determining the manual coding error rate is significant. For this purpose, the error distribution was analyzed using duplicated moments' scores, then MAE and RMSE were calculated as 0.169 and 0.251, respectively.

For the first phase of the training process, 5 different CNN architectures were compared with roughly-defined parameters. For regression type, the cross-validation results of VGG16 and VGG19 were way ahead of the others and VGG16 was slightly better than VGG19 with overall MAE (test loss) of 0.335.

After regression type comparison, these 5 CNN architectures were also compared by means of classification performances. 3-Alternative forced network architecture were created and trained with rough parameters. The winner of the five was observed as VGG16 with overall cross-validation test accuracy of 66.3%.

All the five CNNs were previously trained using the same database, ImageNet. The reason that VGGNets performed far better than the others may be deeper networks are not required for the dataset used in this study. Another reason could be the selected parameters well-suited to the VGGNets by chance. This case has too little possibility as several training iterations were executed with different parameters.

At the initial part of the second phase, the performances of several different network configurations were compared using single cross-validation set. Normalization options (input normalization and batch normalization), the size of the fully connected layers, learning rate, and different activation functions were studied. Although some improvement was achieved by changing hyper-parameters for this single cross-validation set, it was observed that overall cross-validation result did not show any improvement. Then, a different configuration was implemented that the convolutional layers were also trained along with the fully connected layers. A performance increase was observed with this configuration that the overall CV MAE was improved to 0.278 for one of the balanced test cases. This value was further improved to 0.26 by incorporating the multi-camera method. If a single CV set is considered, a minimum MAE of 0.226 was achieved for one of the balanced test cases with multi-camera method.

Even though an improvement was observed for the selected CV dataset by fine tuning parameters, the overall CV results did not improve until the convolutional layers were set as trainable. In fact, this is an indication of dissimilarity between the two datasets, ImageNet and the one used in this study.

Among all the previous studies related to the automation of MGS, regression analysis was not carried out. For this reason, it is not possible to make a comparison in this manner. However, the internal scoring consistency of an expert could be used as a reference of how the network performs. As stated before, each data was scored by only one expert and the average MAE of the manual scoring was calculated as 0.169 using randomly chosen duplicated data. On the other hand, the analysis of inter-scorer reliability was not performed by any means. It is possible that the manual scoring consistency may worsen if multiple scorers were considered. For this reason, achieving an overall CV MAE of 0.26 is quite promising for practical implementations of MGS automation.

Within the four test cases, even though the unbalanced test data presents the best performance, the balanced one should be taken account. This is because a higher performance of an imbalanced dataset may hide its poor performance when faced with a different dataset. Moreover, it is also desired to have the numbers (performances of different test cases) closer to each other, since distant results may be an indication of poor performance specific to some input space.

Considering the full automation of MGS, the results of the original test dataset is more important than that of corrected, because human intervention was required to construct the corrected version of the dataset even it stayed at the lowest level. However, the intervention in this study was limited to the detection results. This means that, it is possible to achieve better results by enhancing the mouse face detection process.

For the regression results, using the multi-camera method improved the performance slightly. Even this slight improvement worth using multi-camera method, this is because it was usually hard or not possible to improve the network performance beyond some value by fine tuning hyper-parameters. Additionally, it is possible that the behavior of the network might prevent the overall result benefiting from the multi-camera method partially. It was observed by analyzing the ME and MAE distributions that the regression networks usually tried to squeeze the outputs into the middle score range which results in biased errors rather than a uniformly distributed errors. As a result, the multi-camera method lacks improving the performance when it faces with biased errors.

The outputs of the regression networks were also discretized in order to obtain binary classification result. Using this method, a maximum overall CV accuracy of 70.75% was achieved with a balanced test dataset for binary classification.

The result of this method ranked at the bottom for binary classification performance. It was also observed that the gap between the results of the balanced dataset and its unbalanced one is quite much which may indicate that one of the classes is poorly handled by the model. The poor performance of this method may be caused by the training process that the cost function of the regression network does not take a higher value when the output slightly exceeds the classification border.

For binary classification network, three different configurations were implemented. In the first one, all the images are used in training without any filtering based on scores. Using this network with multi-camera method, 82% overall CV accuracy was achieved for one of the balanced test cases. Considering a single CV set, a maximum of 91.10% accuracy was observed for the balanced test cases with multi-camera method.

In the second binary network configuration, the images belonging to the score ranges [0.5, 1.0) were discarded. Using this configuration with multi-camera method, 82.45% overall CV accuracy was achieved.

For the last binary network configuration, the medium pain score range, [0.5, 1.5), was discarded while training. With this configuration, 79.53% overall CV accuracy was achieved.

For the binary classification results, the performance improvement using the multi-camera method is much more apparent than that of regression results. An accuracy increase of almost 3% was observed for most of the cases. It is estimated that the increase amount would go up if the overall accuracy without multi-camera method is further improved acting as a positive feedback.

It was observed that, discarding a specific score range for training may improve the overall performance of the classification. With discarding a score range around the classification border, the ambiguous data would have been removed for the classification task. Thus, it would become easier for the network to separate the input space into desired classes.

In the previous study [2], classification was performed for three classes and it was stated that 86% accuracy was achieved with a new test data. Considering the 91.10% binary classification accuracy of a single CV set in this study, the both results could be in a competence. It should be better to note that, in this study, the maturity of the trained network was always decided with a separate validation dataset. Thus, the resulting network was not positively biased against the test data in any circumstances. However, in the previous study [2], validation data was not used and the specifications of the stated test data was not given.

In a later study [5], it was stated that 84% binary classification accuracy was achieved for a dataset that was not used in training. It was also reported that the accuracy was increased to 94% if low-confidence images were discarded from the dataset. On the other hand, none of the cross-validation methods was mentioned in the study. Thus, the binary classification accuracy, 91.10%, achieved for the LOSOCV-5 with multi-camera method can be compared with the 84% accuracy of the study [5] and can be commented as a better result. Moreover, in our study, it was observed that discarding the medium score region from the test dataset provided an increase in the overall CV accuracy up to 96.38% for binary classification. For the same case, if the evaluation is carried out on a single CV dataset, the best achieved binary classification accuracy was observed as 98.8%.

In the latest work [4], some similar methods were adopted as in this study. It was stated that 10-fold cross-validation and leave-one-animal-out-cross validation methods were used. However, the dataset was split into two without having a separate validation dataset in contrast to this study. Multiple frame evaluation was mentioned like the multi-camera method, however, the required details (number of cameras, time accuracy, limitations, etc.) were not given in the paper. On the other hand, constructing and labeling the dataset was handled in a different way such that a minor group of images were manually labeled according to MGS. Then the major part of the dataset was automatically collected and labeled based on the specific time points rather than random time points. In this sense, the suitability of the resulting network for most of the practical implementations would become arguable. The study stated that 98.9% binary classification CV accuracy was achieved for one type of pain stimulation while the second best accuracy was achieved as 90.1% for another pain type. Assuming that similar cross-validation methods were implemented, the achieved binary classification overall CV accuracy in this study, 82.45%, is far behind the results given in that previous study [4]. However, considering the other differences such as dataset, pain stimulation and labeling methods, it would be unfair to state such a clear comparison.

For one of the future works, the effect of the individual scores of the AUs on the prediction performance could be analyzed and the AUs having the worst performance would be discarded. As a second future work, the number of the experiments could be increased in order to totally eliminate the need of data augmentation and if possible, each data would be scored by multiple scorers. Similarly, training data could be increased by synthetically generating intermediate frames using multiple frames of six camcorders. For another future work, particularly for the regression network, the discontinuity problem for the scores of around 0 and 2 could be investigated and possible solutions could be implemented. Besides, mouse face detector and tracker models could be improved by using the ready-to-use data collected within the scope of this study. In addition to those, the network structure could be altered such that multiple face images (belong to same instant) are fed simultaneously as input rather than applying them individually to the network. And as a final future work, an online scoring tool could be developed using the trained regression network in this study.

REFERENCES

[1] D. J. Langford *et al.*, “Coding of facial expressions of pain in the laboratory mouse,” *Nat. Methods*, 2010.

[2] M. Eral, “Deep learning approach for Laboratory mice grimace scaling,” Middle East Technical University, 2016.

[3] M. Eral, C. C. Aktas, E. E. Kocak, T. Dalkara, and U. Halici, “Assessment of pain in mouse facial images,” 2017.

[4] N. Andresen *et al.*, “Towards a fully automated surveillance of well-being status in laboratory mice using deep learning,” *bioRxiv*, 2019.

[5] A. H. Tuttle *et al.*, “A deep neural network to assess spontaneous pain from mouse facial expressions,” *Mol. Pain*, 2018.

[6] J. . Langford, D.J., Bailey, A.L., Chanda, M.L., Clarke, S.E., Drummond, T.E., Echols, S., Glick, S., Ingrao, J., Klassen-Ross, T., LaCroix-Fralish, M.L., Matsumiya, L., Sorge, R.E., Sotocinal, S.G., Tabaka, J.M., Wong, D., van den Maagdenberg, A.M.J.M., Ferra, “Mouse Grimace Scale: The manual,” *Nat. Methods*, 1920.

[7] S. G. Sotocinal *et al.*, “The Rat Grimace Scale: A partially automated method for quantifying pain in the laboratory rat via facial expressions,” *Mol. Pain*, 2011.

[8] E. Dalla Costa, M. Minero, D. Lebelt, D. Stucke, E. Canali, and M. C. Leach, “Development of the Horse Grimace Scale (HGS) as a pain assessment tool in horses undergoing routine castration,” *PLoS One*, 2014.

[9] A. V. Viscardi, M. Hunniford, P. Lawlis, M. Leach, and P. V. Turner, “Development of a Piglet Grimace Scale to Evaluate Piglet Pain Using Facial Expressions Following Castration and Tail Docking: A Pilot Study,” *Front. Vet. Sci.*, 2017.

[10] C. Häger *et al.*, “The Sheep Grimace Scale as an indicator of post-operative distress and pain in laboratory sheep,” *PLoS One*, 2017.

[11] B. Akkaya, Y. R. Tabar, F. Gharbalchi, İ. Ulusoy, and U. Halıcı, “Tracking mice face in video,” in *2016 20th National Biomedical Engineering Meeting (BIYOMUT)*, 2016, pp. 1–4.

[12] İ. B. Akkaya, “Mouse face tracking using convolutional neural networks,” Middle East Technical University, 2016.

[13] C. Darwin and F. Darwin, *The expression of the emotions in man and animals*.

2009.

[14] C.-H. Hjortsjö, *Man's face and mimic language*. Lund: student literature, 1970.

[15] P. Ekman and W. V. Friesen, *The Facial Action Coding System.* 1978.

[16] K. J. Sufka, "Conditioned place preference paradigm: a novel approach for analgesic drug assessment against chronic pain," *Pain*, 1994.

[17] K. A. Grimm, L. A. Lamont, W. J. Tranquilli, S. A. Greene, and S. A. Robertson, *Veterinary Anesthesia and Analgesia: The Fifth Edition of Lumb and Jones*. 2015.

[18] F. C. Colpaert, J. P. Tarayre, M. Alliaga, L. A. Bruins Slot, N. Attal, and W. Koek, "Opiate self-administration as a measure of chronic nociceptive pain in arthritic rats," *Pain*, 2001.

[19] J. V. Roughan and P. A. Flecknell, "Evaluation of a short duration behaviour-based post-operative pain scoring system in rats," *Eur. J. Pain*, 2003.

[20] A. L. Miller and M. C. Leach, "The mouse grimace scale: A clinically useful tool?," *PLoS One*, 2015.

[21] M. Leach, "Rabbit Grimace Scale (RbtGS) Manual," *PLoS One*, 2012.

[22] S. A. Papert, "The summer vision project," 1966.

[23] R. A. Brooks, R. Creiner, and T. O. Binford, "The ACRONYM Model-based Vision System," in *Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 1*, 1979, pp. 105–113.

[24] M. A. Fischler and R. A. Elschlager, "The Representation and Matching of Pictorial Structures," *IEEE Trans. Comput.*, vol. 22, no. 1, pp. 67–92, 1973.

[25] D. G. Lowe, "Three-dimensional object recognition from single two-dimensional images," *Artif. Intell.*, 1987.

[26] P. Viola and M. J. Jones, "Robust Real-time Object Detection," 2001.

[27] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *ImageNet Classification with Deep Convolutional Neural Networks*, 2012.

[29] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1, 2014.

[30] C. Szegedy *et al.*, "GoogLeNet Going Deeper with Convolutions," *arXiv Prepr.*

arXiv1409.4842, 2014.

[31] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[32] M. Monwar and S. Rezaei, “Pain Recognition Using Artificial Neural Network,” 2006, pp. 28–33.

[33] G. Littlewort, M. Stewart Bartlett, and K. Lee, “Faces of pain: automated measurement of spontaneousallfacial expressions of genuine and posed pain,” in *Proceedings of the International Conference on Multimodal Interfaces*, 2007, pp. 15–21.

[34] A. Ashraf *et al.*, “The Painful Face - Pain Expression Recognition Using Active Appearance Models,” *Image Vis. Comput.*, vol. 27, pp. 1788–1796, 2009.

[35] Z. Hammal and M. Kunz, “Pain monitoring: A dynamic and context-sensitive system,” *Pattern Recognit.*, vol. 45, pp. 1265–1280, 2012.

[36] J. Zhou, X. Hong, F. Su, and G. Zhao, “Recurrent Convolutional Neural Network Regression for Continuous Pain Intensity Estimation in Video,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2016.

[37] M. Tavakolian and A. Hadid, “A Spatiotemporal Convolutional Neural Network for Automatic Pain Intensity Estimation from Facial Dynamics,” *Int. J. Comput. Vis.*, 2019.

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[39] G. M. Foody, “The effect of mis-labeled training data on the accuracy of supervised image classification by SVM,” in *2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, 2015, pp. 4987–4990.

[40] E. Lebeau, “Study of label errors on a convolutional neural network,” UCL, 2017.

[41] “Biological to Artificial Neuron.” [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/4/44/Neuron3.png. [Accessed: 04-Jul-2019].

[42] J. Duchi, E. Hazan, and Y. Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” *JMLR*, 2011.

[43] T. Tieleman, G. E. Hinton, N. Srivastava, and K. Swersky, “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude,”

COURSERA Neural Networks Mach. Learn., 2012.

[44] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *Int. Conf. Learn. Represent.*, 2014.

[45] D. H. Hubel and T. N. Wiesel, “Receptive fields and functional architecture of monkey striate cortex.,” *J. Physiol.*, 1968.

[46] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, 1998.

[47] S. E. Clarke *et al.*, “Coding of facial expressions of pain in the laboratory mouse,” *Nat. Methods*, 2010.

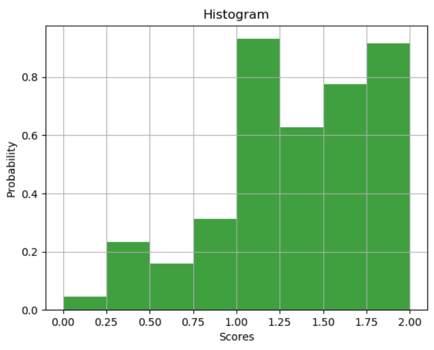
[48] C. J. Willmott and K. Matsuura, “Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance,” *Clim. Res.*, 2005.

[49] C. J. Willmott, K. Matsuura, and S. M. Robeson, “Ambiguities inherent in sums-of-squares-based error statistics,” *Atmos. Environ.*, 2009.

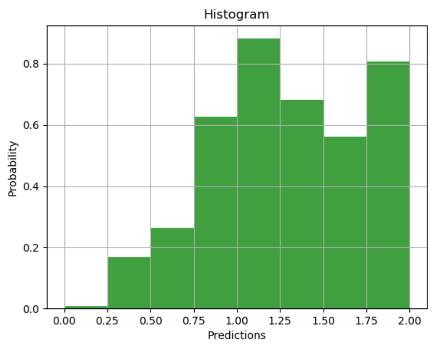
[50] T. Chai and R. R. Draxler, “Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature,” *Geosci. Model Dev.*, vol. 7, no. 3, pp. 1247–1250, 2014.

APPENDICES

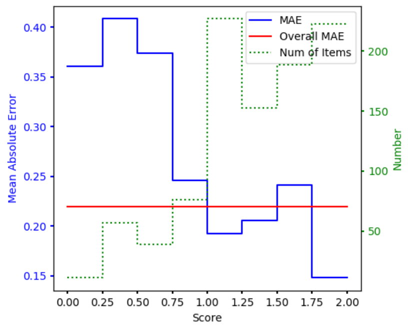
A. TEST-CORR Multi-Camera Dataset and the Network, $N_R^{(3)}$, Prediction Statistics



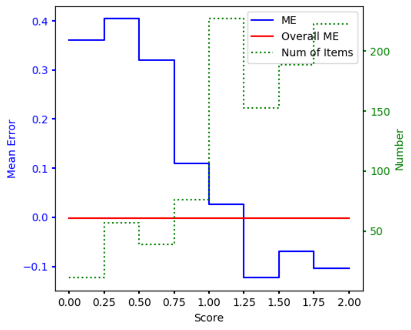
(a)



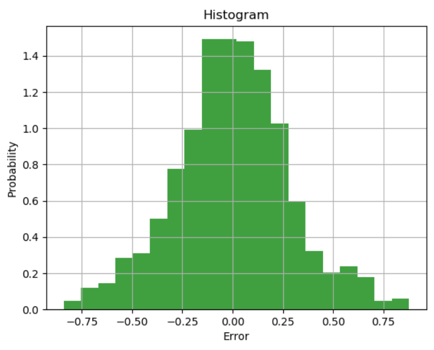
(b)



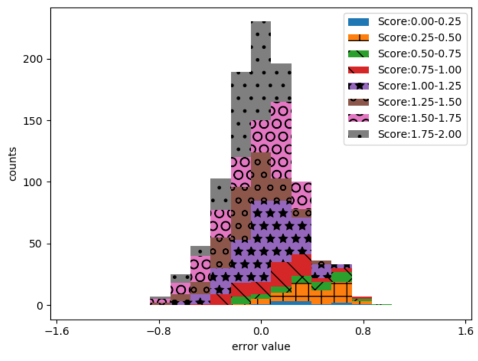
(c)



(d)



(e)



(f)

Figure A.1. $N_R^{(3)}$ LOSOCV-1 TEST-CORR Multi-Camera Statistics; (a) Moment Score Distribution, (b) Moment Prediction Distribution, (c) MAE vs Score Groups and Number of Moments Distribution, (d) ME vs Score Groups and Number of Moments Distribution, (e) Error Distribution, (f) Error Distribution with Score Groups

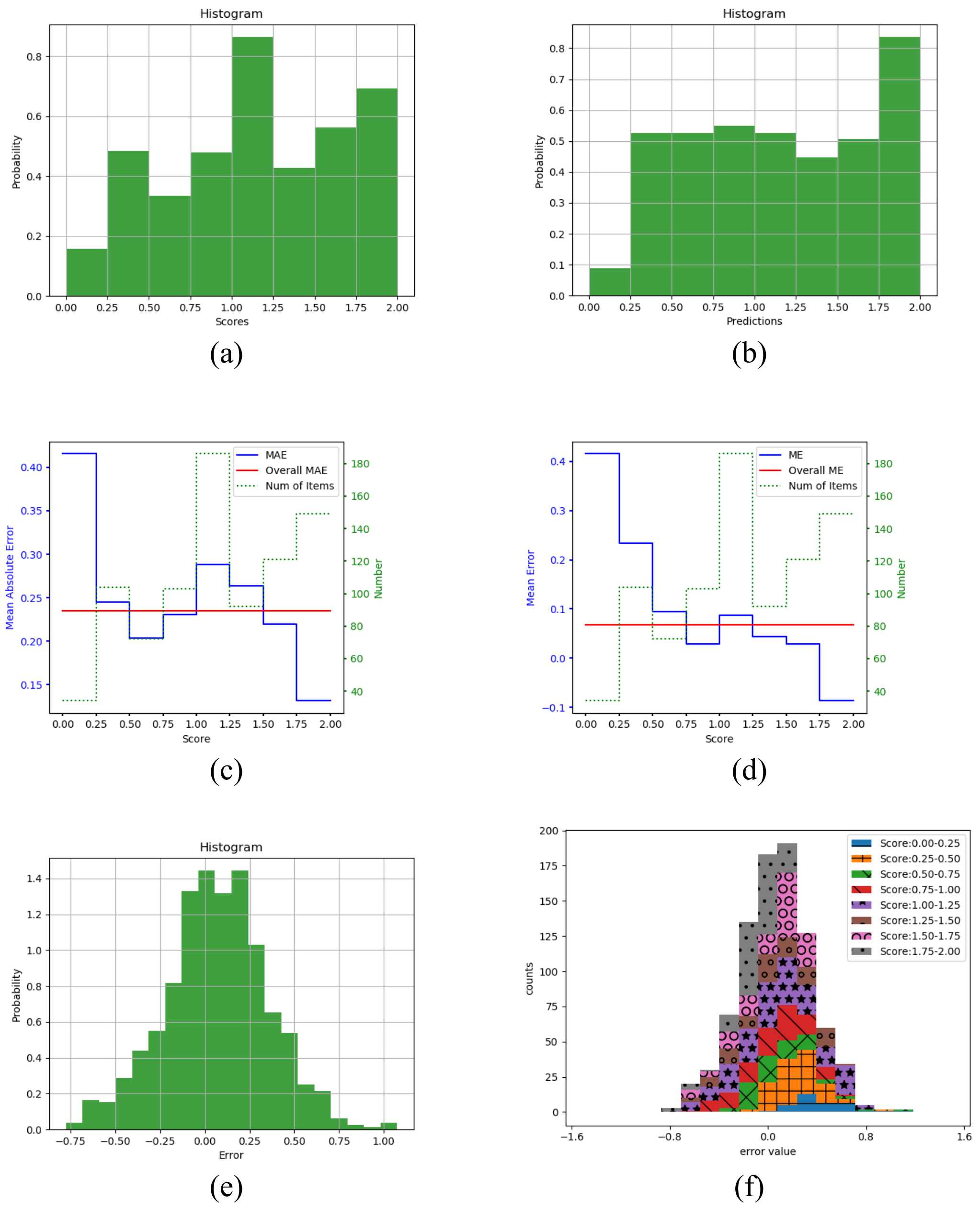
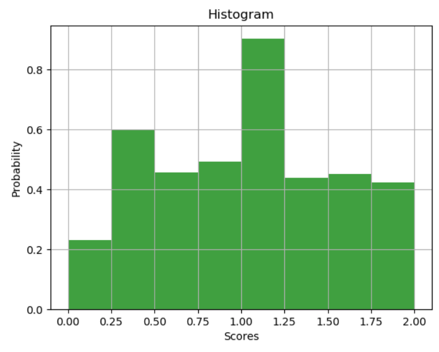
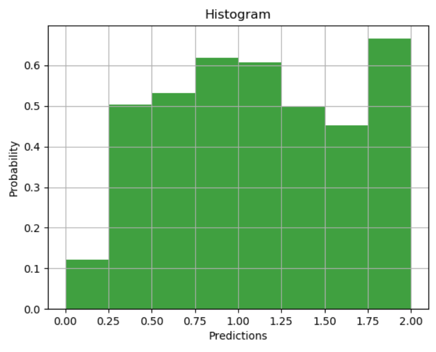


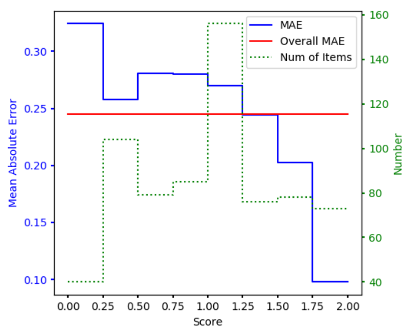
Figure A.2. $N_R^{(3)}$ LOSOCV-2 TEST-CORR Multi-Camera Statistics; (a) Moment Score Distribution, (b) Moment Prediction Distribution, (c) MAE vs Score Groups and Number of Moments Distribution, (d) ME vs Score Groups and Number of Moments Distribution, (e) Error Distribution, (f) Error Distribution with Score Groups



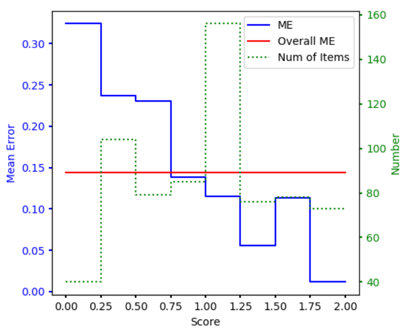
(a)



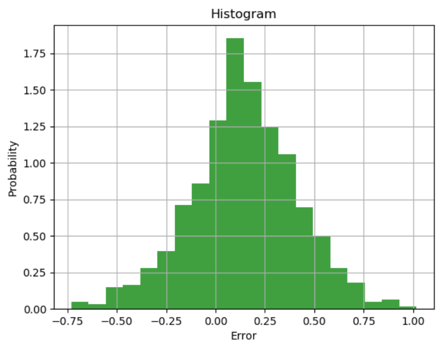
(b)



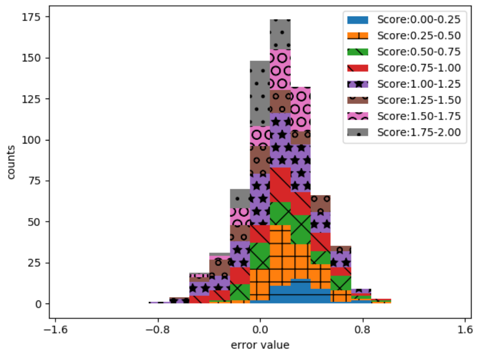
(c)



(d)

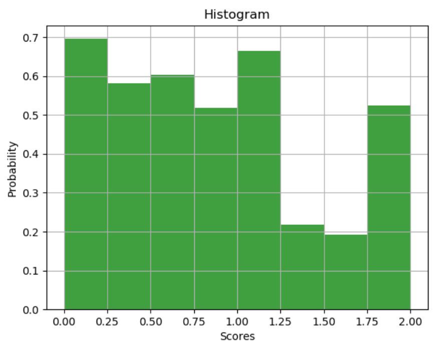


(e)

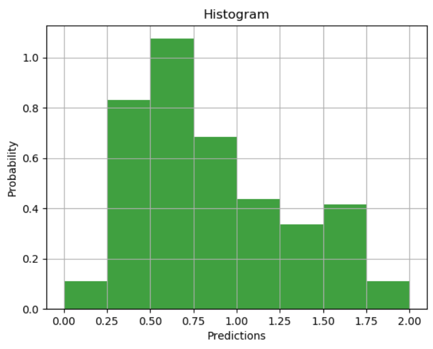


(f)

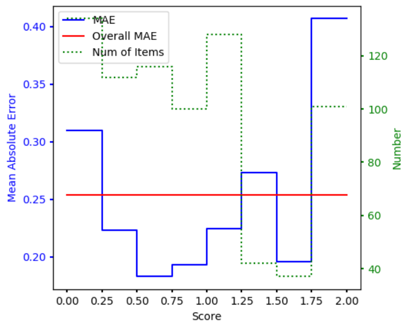
Figure A.3. $N_R^{(3)}$ LOSOCV-3 TEST-CORR Multi-Camera Statistics; (a) Moment Score Distribution, (b) Moment Prediction Distribution, (c) MAE vs Score Groups and Number of Moments Distribution, (d) ME vs Score Groups and Number of Moments Distribution, (e) Error Distribution, (f) Error Distribution with Score Groups



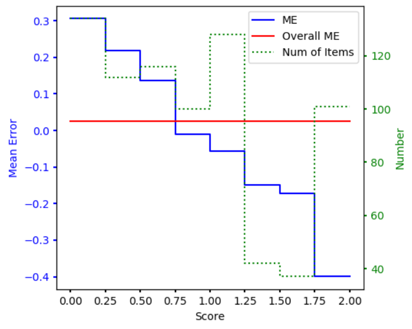
(a)



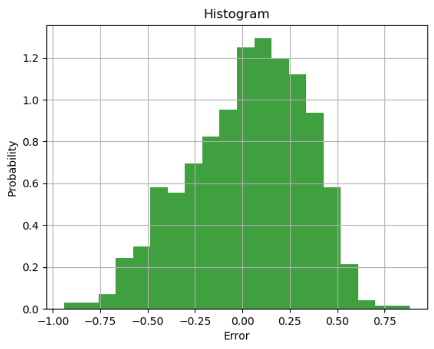
(b)



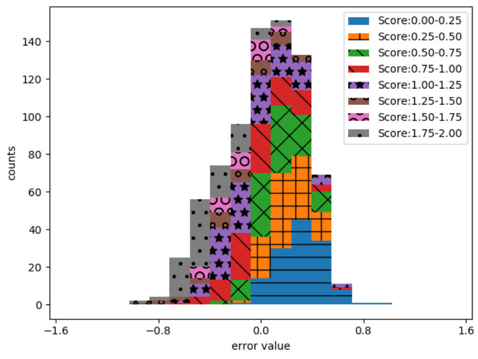
(c)



(d)

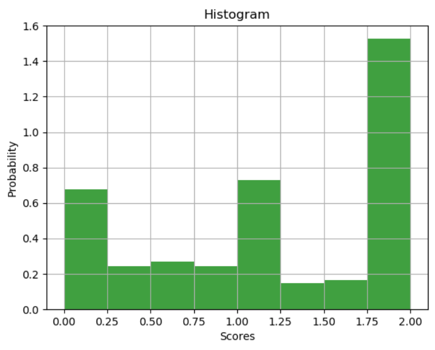


(e)

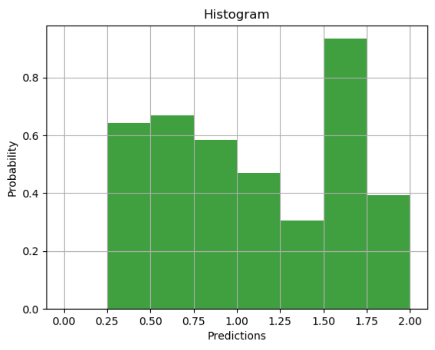


(f)

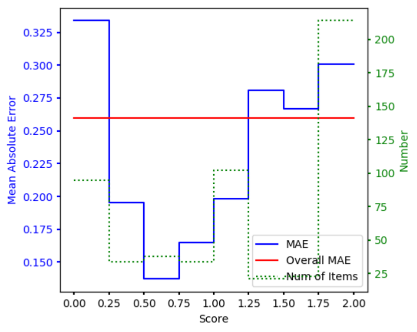
Figure A.4. $N_R^{(3)}$ LOSOCV-4 TEST-CORR Multi-Camera Statistics; (a) Moment Score Distribution, (b) Moment Prediction Distribution, (c) MAE vs Score Groups and Number of Moments Distribution, (d) ME vs Score Groups and Number of Moments Distribution, (e) Error Distribution, (f) Error Distribution with Score Groups



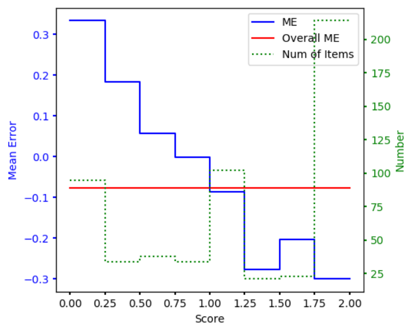
(a)



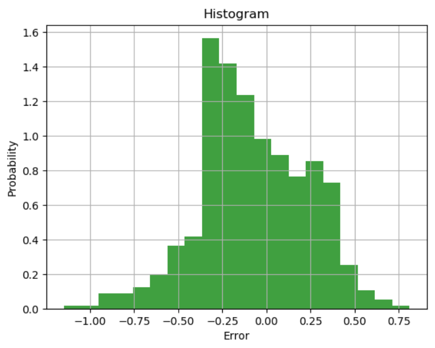
(b)



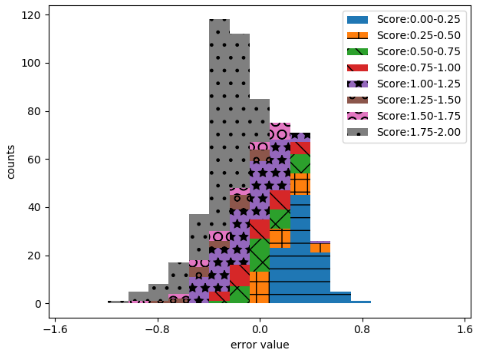
(c)



(d)

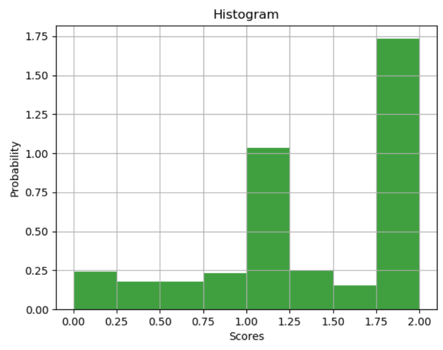


(e)

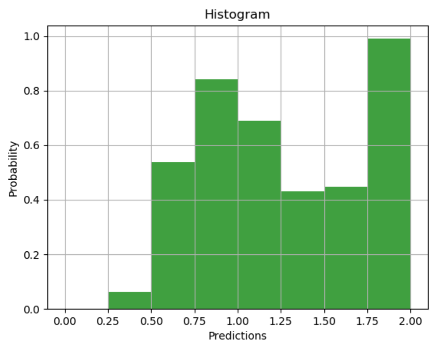


(f)

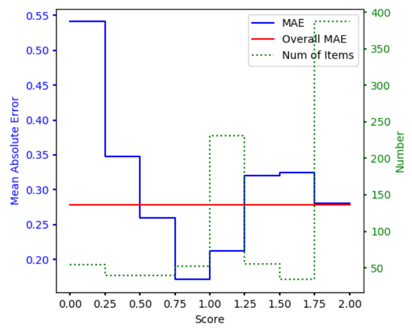
Figure A.5. $N_R^{(3)}$ LOSOCV-5 TEST-CORR Multi-Camera Statistics; (a) Moment Score Distribution, (b) Moment Prediction Distribution, (c) MAE vs Score Groups and Number of Moments Distribution, (d) ME vs Score Groups and Number of Moments Distribution, (e) Error Distribution, (f) Error Distribution with Score Groups



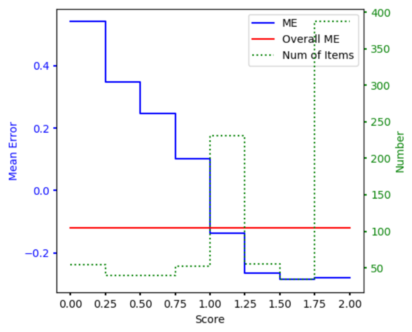
(a)



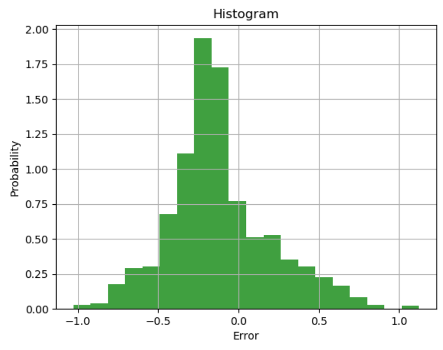
(b)



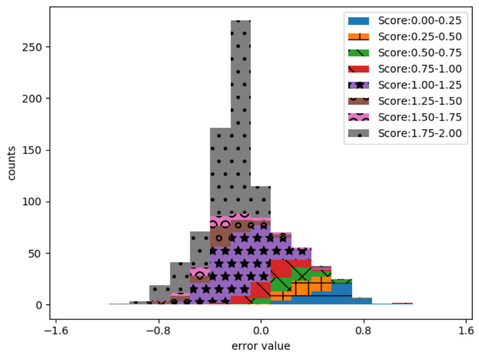
(c)



(d)



(e)



(f)

Figure A.6. $N_R^{(3)}$ LOSOCV-6 TEST-CORR Multi-Camera Statistics; (a) Moment Score Distribution, (b) Moment Prediction Distribution, (c) MAE vs Score Groups and Number of Moments Distribution, (d) ME vs Score Groups and Number of Moments Distribution, (e) Error Distribution, (f) Error Distribution with Score Group