AN ENHANCEMENT OF AGE AND GENDER CLASSIFICATION ACCURACY WITH HYBRID HANDCRAFTED AND DEEP FEATURES USING HIERARCHICAL EXTREME LEARNING MACHINE

MOHAMMAD JAVIDAN DARUGAR

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2020

AN ENHANCEMENT OF AGE AND GENDER CLASSIFICATION ACCURACY WITH HYBRID HANDCRAFTED AND DEEP FEATURES USING HIERARCHICAL EXTREME LEARNING MACHINE

MOHAMMAD JAVIDAN DARUGAR

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2020

UNIVERSITI MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: **Mohammad Javidan Darugar**

Registration/Matrix No.: **Old: WGA140037 | New: 17049741/1**

Name of Degree: **Master of Computer Science**

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

An enhancement of Age and Gender classification accuracy with hybrid handcrafted and deep features using Hierarchical ELM

Field of Study: **Artificial Intelligence**

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date

Subscribed and solemnly declared before,

Witness's Signature Date

Name:
Designation:

ABSTRACT

Age and gender classification are some of the essential algorithms that have many use cases in our everyday life. For example, in robotics, field robots can interact with a human base on their gender in data analysis, to have statistics about age and gender of audiences in social events, YouTube video analysis, and many other applications. In this research, we have addressed limitations in deep neural networks, which by overcoming this limitation, we can gain better accuracy and performance. Our study has several other possible applications which are not limited only to age and gender classification. This dissertation is about a high-performance method for age and gender classification in captured facial photos, which is employing deep network architectures as the primary basis of our architecture. We have employed branches of deep learning, such as convolutional neural networks and autoencoders. We have also used Hierarchical Extreme Learning Machines to avoid significant time consumption and to overcome limitations that most conventional deep networks have. We have addressed the problem of using Softmax as the classifier, which usually leads to less performance and accuracy due to its limitation, which is only able to classify linearly separable data. Our proposed architecture consists of two main categories of feature extraction and learning methods which are namely supervised and unsupervised. We have investigated two supervised feature extraction techniques and a deep feature extraction technique to extract unsupervised features to judge the influence of each one on the accuracy of our proposed model. The supervised methods that are examined in this study are Histogram of Oriented Gradients or HOG and Action Units (or AUs). For unsupervised feature extraction techniques, we have employed a deep neural network which is known as convolutional neural network (CNN). CNNs use shift-invariant filters to make discriminative features inside neural networks. One of the very potent tools

is convolutional neural networks, which is used for obtaining useful features that also are useful for unknown classes. In our research, we figured out that some of our features are high, and some are very low in dimension. So to combine these groups of supervised and unsupervised features with different dimensions, we have used multiple autoencoder neural networks to join, reduce, and encode all employed feature maps into a single feature vector. Hierarchical-ELM, which is a branch of Extreme Learning Machines, is adopted to classify the final feature vector. Toward this research, we have analyzed the result of our proposed work with state of the art, and also related works are explained to illustrate our significant improvement for age and gender classification in facial images. Our gains are in both accuracy and performance. Regarding performance, we have achieved faster training and testing process. Since we are dealing with large datasets of facial images, therefore, speeding up these steps can influence a more reliable solution.

Keywords: Deep Neural Networks, Age and Gender Classification, Hierarchical Extreme Learning Machine, Convolutional Neural Network.

ABSTRAK

Klasifikasi umur dan jantina adalah beberapa algoritma penting yang mempunyai banyak kes penggunaan dalam kehidupan seharian kita. Contohnya, dalam bidang robotik, robot bidang boleh berinteraksi dengan asas manusia mengenai jantina mereka dalam analisis data, untuk mempunyai statistik mengenai Umur dan Gender khalayak dalam acara sosial, analisis video YouTube, dan banyak aplikasi lain. Dalam penyelidikan ini, kita telah menangani batasan-batasan dalam rangkaian saraf yang mendalam, yang dengan mengatasi had ini, kita dapat memperoleh ketepatan dan prestasi yang lebih baik. Kajian kami mempunyai beberapa aplikasi lain yang mungkin tidak hanya terhad kepada klasifikasi umur dan jantina. Disertasi ini adalah tentang kaedah berprestasi tinggi untuk klasifikasi Umur dan Jantina dalam foto wajah yang diambil, yang menggunakan seni bina rangkaian dalam sebagai asas utama seni bina kami. Kami telah mempekerjakan cabang-cabang pembelajaran yang mendalam, seperti convolutional neural network dan autoencoder. Kami juga telah menggunakan Mesin Pembelajaran Extreme Hierarki untuk mengelakkan penggunaan masa yang ketara dan untuk mengatasi batasan-batasan yang mempunyai rangkaian dalam yang paling konvensional. Kami telah menangani masalah menggunakan Softmax sebagai pengelas, yang biasanya membawa kepada kurang prestasi dan ketepatan kerana batasannya, yang hanya dapat mengklasifikasikan data yang boleh diasingkan secara linear. Senibina kami yang dicadangkan terdiri daripada dua kategori utama pengekstrakan ciri dan kaedah pembelajaran yang diawasi dan tidak diselia. Kami telah menyiasat dua teknik pengekstrakan ciri yang diawasi dan teknik pengekstrakan ciri yang mendalam untuk mengekstrak ciri-ciri yang tidak diselia untuk menilai pengaruh masing-masing berdasarkan ketepatan klasifikasi kami. Kaedah yang diawasi dalam kajian ini adalah Histogram Gradients Oriented atau HOG, Action Units atau AU. Untuk

teknik ekstraksi ciri yang tidak terjejas, kami telah menggunakan rangkaian saraf yang mendalam yang dikenali sebagai convolutional neural network (CNN). CNNs menggunakan penapis peralihan bergerak untuk membuat ciri diskriminatif di dalam rangkaian saraf. Salah satu alat yang sangat kuat ialah convolutional neural network, yang digunakan untuk mendapatkan ciri-ciri berguna yang juga berguna untuk kelas yang tidak diketahui. Dalam penyelidikan kami, kami mendapati bahawa beberapa ciri kami adalah tinggi, dan ada yang sangat rendah dalam dimensi. Jadi, untuk menggabungkan kumpulan ciri-ciri yang diselia dan tidak diselia dengan dimensi yang berbeza, kami telah menggunakan pelbagai autoencoder neural network untuk menyertai, mengurangkan, dan mengodkan semua peta ciri yang digunakan ke dalam vektor ciri tunggal. Hierarki-ELM, yang merupakan cabang Mesin Pembesaran Extreme, digunakan untuk mengklasifikasikan vektor ciri akhir. Dalam kaji selidik ini, kami telah menganalisis hasil karya yang dicadangkan kami dengan keadaan seni, dan juga kerja-kerja yang berkaitan dijelaskan untuk menggambarkan peningkatan yang signifikan untuk klasifikasi Umur dan Jantina dalam imej muka. Keuntungan kami berada dalam kedua-dua ketepatan dan prestasi. Mengenai prestasi, kami telah mencapai proses latihan dan ujian yang lebih cepat. Oleh kerana kita berhadapan dengan kumpulan data wajah yang besar, oleh itu, mempercepatkan langkah-langkah ini dapat mempengaruhi penyelesaian yang lebih dapat diandalkan.

Keywords: Deep Neural Networks, Age and Gender Classification, Hierarchical Extreme Learning Machine, Convolutional Neural Network.

ACKNOWLEDGEMENTS

First, I want to give my genuine appreciation to my advisor and guidance Prof. Dr. Loo Chu Kiong for all constant assistance and help for my Master study. For all of my passion and success during my study that created by his motivation and patience, and vast experience and expertise. His supervision encouraged me during my study and research and working on this dissertation. Dr. Aznul Qalid Bin Md Sabri, Dr. Unaizah Hanum Binti Obaidellah, and Dr. Woo Chaw Seng, I would like to thank you for your motivating, discerning remarks, and challenging questions. Moreover, I want to give my most profound love to my mother Maryam Chapari Rasi for raising me and supporting me during my whole life, and to my wife Anahita for supporting me during my study as my best friend and accompanying me in all hardships in throughout my life. I want to thank one of the best couples I have known during my study in Malaysia, dear Sina and Fatima, thanks for all your helps and supports.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1: INTRODUCTION

1.1 Background

Age and gender are becoming valuable information in data science, and also play essential roles in social interactions. Concerning data science, information like age and gender can be used in recommendation systems or regarding robotics for robots that interact with Human languages have various greetings and grammar rules for each gender. For instance, typically separate word lists are employed while speaking with seniors compared to juveniles. Early designs for age and gender classification are employing supervised or unsupervised features classification. For example, supervised features like facial texture using Local Binary Pattern or 3D structure of the head and unsupervised features like features that are extracted using deep neural networks like convolutional neural network which will be explained in more detail in next chapter. In this chapter, we are introducing new research that can improve age and gender tagging or classification in terms of accuracy and performance. In studies that deal with massive datasets time is essential especially for the training process. The main reason behind that is it can lead to unreliable results or may take so much time to gain suitable results in the other hand a fast training process can help to speed up research and achieve proper results faster.

Many data scientists struggle with using huge datasets which means they have to spend hours and use many computational resources to train their models. To be more precise, if we have thousands or even millions of facial images batches that they are used to adjusting weights in the neural network in each iteration, that would be a massive burden for computational resources and time-consuming process because this iteration may happen for many times to converge the best possible solution. We know that in some cases machine learning engineer needs to stop the learning process due to issues,

for example, over-fitting and change some parameters and start over training again. But with the iterative solution, it won't be easy.

Base on our pre-research investigation, most of the researches is either using supervised or unsupervised feature learning methods individually but just a limited number of researches have used a hybrid form of supervised and unsupervised features.

1.1.1 Deep Neural Networks and Softmax

Generally speaking, deep neural networks (DNN) is one of many varieties of artificial neural networks (ANNs) with several hidden layers connecting the input layer to the output layer (Bengio et al., 2009; Schmidhuber, 2015). In DNN designs, the data, e.g., an image is represented as a layered composition of that data in this example image primitives (Szegedy, Toshev, & Erhan, 2013) which helps to have multiple levels of feature abstraction. In other words, it helps to have a composition of features in a bottom-up form of abstraction levels, e.g., from patches to meaningful features such as Cars, Human or Cat face, which helps to ignore redundant features and to model complex data (Bengio et al., 2009). One of the essential modules of Supervised DNNs is the classifier. The commonly used classifier for DNNs is Softmax which implies the generalized form of logistic regression in which one of its properties is to be utilized to describe a distribution in categorical form. It is a likelihood or a distribution of probability across K various potential values. Concerning logistic regression, it is possible to have the target values only in binary: $y^{(i)} \in \{0, 1\}$. For example, the binary form is employed for recognizing data with two classes. e.g., whether positive or negative handwritten digits while on the other hand Softmax regression can supervise $y^{(i)} \in \{0, \ldots, K-1\}$ which if for example if $K = 10$ then it is possible to classify numbers 0 to 9 since there are 10 classes. Considering dataset of $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ with m labeled data, with $x^{(i)} \in \mathfrak{R}^n$ as input feature logistic regression can classify the target values $y^{(i)} \in \{0, 1\}$

which represent binary labels. (Ma, Lu, Zhang, & Tang, 2014).

By assuming $h_\theta(x)$ as the hypothesis we have:

$$h_\theta(x) = \frac{1}{1 + \exp(-\theta^\top x)} \tag{1.1}$$

moreover, considering the loss function to be minimized with paramere θ:

$$J(\theta) = -\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] \tag{1.2}$$

Softmax regression in opposed to binary form that we have discussed, can have label y in K sapace. So, in $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ can be training set which target classes are $y^{(i)} \in \{0, 1, \ldots, K-1\}$.

For an evaluation input data x, to compute the likelihood that $P(y = k|x)$ toward every value of $k = 0, \ldots, K-1$. The probability of the class number is estimated having every k different possible amount. Therefore, it produces an output vector with a K dimension where elements sum is 1.0 providing us K predicted probability values.

Then if we assuem $h_\theta(x)$ is:

$$h_\theta(x) = \begin{bmatrix} P(y = 1|x; \theta) \\ P(y = 2|x; \theta) \\ \vdots \\ P(y = K|x; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix} \tag{1.3}$$

here model parameters are shown as $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)} \in \Re^n$ (Ma et al., 2014).

1.2 Problem Statement

We have noticed that those architectures are using Softmax suffer from lack of accuracy to classify age and gender which caused by limitations of Softmax (Table 1.1).

Table 1.1: Best accuracy of other works

Author	Gender	Age
Dantcheva, A., and Brémond, F. (2016)	75.10%	-
Levi, G., and Hassner, T. (2015).	86.80%	84.70%
E Eidinger, R Enbar, T Hassner (2014)	77.80%	79.50%

The main limitation of the Softmax classifier is that the classification process will not work if the input data is not linearly separable and one more limitation of the Softmax classifier is that it does not support the null rejection. Therefore it is obligated to train the algorithm with a particular null class. Identified research problems are listed as below:

- The first problem is that deep network feature extraction cannot guarantee linear separable features.
- The second problem is that early methods are using Softmax for classifier which Softmax has limitations, the primary and essential limitation of the Softmax algorithm is that it will not work if the provided data is not linearly separable.
- The third problem is that Softmax architectures use iterative training algorithms which leads to performance issues such as slow convergence rate and local minima.
- The fourth problem is that although unsupervised feature learning methods can learn to extract useful features, there is no guarantee that we can extract well-known useful handcrafted features as well.
- The fifth problem is that supervised and unsupervised features can be in very different dimensions which makes the fusion of these groups of features so important because a weak approach to fuse them may lead to an underfitting learning state.

Our **hypothesis** is included of three main conditions:

- To overcome limitations of Softmax that are explained earlier we use a non-linear classifier

- To use a non-iterative training algorithm like a form of Extreme Learning Machines to overcome the performance issue
- A hybrid form of supervised and unsupervised feature learning can lead to a selection of more useful features

1.3 Research Objectives

Considering the problem statement and our hypothesis which are explained in the previous section, we are establishing a research to fulfill our goal to overcome the mentioned problems. Objectives of our research are listed below:

- To use Hierarchical ELM architecture as a non-linear classifier to overcome the earlier mentioned limitation of Softmax which does not need the data to be necessarily linearly separable.
- By choosing Hierarchical ELM as the classifier, we overcome slow convergence rate and local minima.
- To use a fusion of well-known handcrafted features namely Action Units, Histogram of Oriented Gradients as supervised features and a group of convolutional layers of a pre-trained convolutional neural network as an unsupervised feature extraction method.
- To use three autoencoders to fuse Action Units, Histogram of Oriented Gradients and features from convolutional layers.

Therefore our primary goal in this study is to design and implement an architecture to extract a hybrid fusion of handcrafted features namely Action Units and Histogram of Oriented Gradients with deep CNN features and then classify them with Hierarchical ELM as a non-linear classifier for age and gender classification.

1.4 Research Questions

- Can hybrid handcrafted and deep features improve the accuracy of age and gender classification?
- Can be replacing the Softmax classifier with H-ELM leads to any improvement in the accuracy and performance in terms of training time?

1.5 Proposed Method

The proposed method implements an architecture that consists of independent layers, which are namely layer zero, supervised and unsupervised features extraction layer, supervised and unsupervised features fusion and dimensionality reduction layer and a classification layer.

The initial layer or layer zero is our input layer to the whole system. It is an RGB facial image. This layer will be fed to the next and first essential layer of our architecture.

In supervised and unsupervised features extraction layer important features of facial images are extracted. In this research, we want to analyze a hybrid feature learning model, which consists of a hybrid handcrafted and deep features extraction model to extract the most useful features. Histogram Of Gradients and Facial Action Units are a well-known example of facial handcrafted features that we are using in our research. We have used a pre-trained convolutional neural network to extract unsupervised deep features.

In supervised and unsupervised features fusion and dimensionality reduction layer we are trying to combine outputs from feature extraction units in the feature extraction layer. We append HOG, AUs, and CNN features using three autoencoders that they encode features into three features vectors with the same size and then they represent a single vector when they are appended to end of each other respectively.

In H-ELM layer generated feature vector is used as input to the H-ELM classifier to

classify age and gender.

1.6 Contributions of the Study

The contribution of this research is summarized as follow:

- One of the essential contributions of this study is to propose an architecture to learn and extract hybrid handcrafted and deep features. So Histogram Of Gradients and Facial Action Units are chosen for handcrafted feature extraction method and few convolutional layers of a pre-trained convolutional neural network are chosen as a deep feature extraction method. These features are then encoded and joined to each other as the input for the classifier.

- Another important contribution of this study is to use Hierarchical Extreme Learning Machine (H-ELM) as a non-linear classifier to improve the accuracy and also to speed up the training process so the proposed architecture can run considerably faster.

1.7 Dissertation Structure

The rest of this dissertation is structured as below:

- In chapter two we will explain some of the related works, literature review and background of important concepts that are used in this research. We will discuss briefly the key differences between Softmax and H-ELM and how they work.

- In chapter three we will explain our proposed architecture in detail. We will explain how hybrid handcrafted features and deep features are generated through the feature extraction layer, the way they are encoded and how it is prepared to be classified by H-ELM.

- In chapter four we will explain our experiments on well-known and benchmark datasets.
- In chapter five we will illustrate our results gained that we from our experiments from chapter four.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

There are many machine learning models to classify age and gender. These models can either be implemented with a deep or shallow neural network. For example (van de Wolfshaar, Karaaba, & Wiering, 2015), and (Levi & Hassner, 2015) use CNN or convolutional neural networks to extract facial features to classify age and gender. (Eidinger, Enbar, & Hassner, 2014) on the other hand, uses Local Binary Pattern or LBP and FPLBP alongside a Support Vector Machine or SVM classifier for age and gender classification. Some techniques use different training set from the testing set for benchmark evaluation. As an instance, (R. Rothe, Timofte, & Gool, 2016) uses the WIKI-IMDB dataset as the training set and Adience as the testing set, which leads to higher accuracy. Here in this chapter, we explain important concepts that are used in our research.

2.1 Deep Artificial Neural Networks

A deep neural network or DNN designs form a configuration where the object is denoted as compositions of image primitives in multiple layers (Szegedy et al., 2013). These extra layers that exist in deep neural networks help to represent features from a deficient level, which can perform similarly to shallow networks but with much fewer units to model complex data (Bengio et al., 2009). Convolutional deep neural networks or CNNs are utilized vastly in computer vision (LeCun, Bottou, Bengio, & Haffner, 1998). CNNs have shown excellent results to model ASR (Sainath, Mohamed, Kingsbury, & Ramabhadran, 2013).

We can train a CNN by one of the conventional delta-rule based algorithms. Backpropagation (or BP) comprises an algorithm to compute the gradient of the loss function concerning the weights in an artificial neural network. The loss function in backpropagation computes the contrast among the training set input data and target output after the

data has gone through the network. Backpropagation is regularly employed to optimize the network concerning the performance by altering the weights.

It mainly consists of two essential stages: 1) propagation and 2) weight update. The given input vector to the network will be fed forward into the next layers continuously until it reaches the output layer. The values that are present in the output layer are analyzed against the actual target output. The loss function will help to measure error value for each neuron in that layer. The propagation of computed error values from the output to backward is the step in which every neuron obtains associated error amount, which approximately describes its contribution to the actual target value.

BP employs computed errors to measure the gradient of the loss function. Then the measured gradient is used by the optimization process utilizes it to adjust the weights to lessen the loss function. The mentioned step is essential since the network has been trained, and the hidden units are adjusted, so each neuron learns to recognize various features concerning the entire input range.

Later, once the training process is done, whenever an arbitrary input data is fed to the input layer, which brings added unknown features, hidden units will react by an active output if the current input data contain a pattern that matches a feature that the single neurons have trained to identify.

BP needs a recognized, suitable output for every input value to compute the loss function gradient. Therefore, it is mostly known and used as a supervised learning technique, although it is applied to unsupervised learning methods such as autoencoders.

Common problems that are known for most multi-layer neural networks are computation time and overfitting. It is widespread for DNNs to overfit, and the reason is that the layers of abstraction allow the network to model rare dependencies in the training data. Commonly, regularization techniques Ivakhnenko's unit pruning (Ivakhnenko,

1971), weight decay ℓ_2 or sparsity ℓ_1 are examples that can be practiced while training to overcome overfitting (Bengio, Boulanger-Lewandowski, & Pascanu, 2013). Alternatively, dropout performs very well concerning regularization, which randomly drops units based on a probability value from the hidden units while training (Dahl, Sainath, & Hinton, 2013).

DNNs rely on several training parameters, namely the layers depth, the capacity of each layer regarding processing units, initial weights, and, last but not least, the learning rate. Finding the most optimal parameters is not strait-forward, and it usually needs lots of experiments and computations, which costs time. There are ways to escape from this bottleneck which, as an instance, it is common to use batches of the training set to speed up computation (Hinton, 2010). GPU processing has also introduced considerable speedups in training since GPUs are excellent tools for matrix-based computations (Schmidhuber, 2015). Most popular alternatives to iterative solutions such as BP are "Extreme Learning Machines" (G.-B. Huang, Zhu, & Siew, 2006) training without backtracking (Ollivier, Tallec, & Charpiat, 2015) "No-prop" networks,(Widrow, Greenblatt, Kim, & Park, 2013) "weightless networks" (Aleksander, De Gregorio, França, Lima, & Morton, 2009) and last but not least non-connectionist neural networks (Robinson, 1994). In further sections, Extreme Learning Machines will be explained in detail.

Loss Functions: Loss or Cost function is an essential part of a broad neural network. Loss functions same as linear or other parametric models represent a distribution $p(y \mid x; \theta)$, and which merely uses the policy of highest probability which indicates that the cross-entropy as the loss function is an excellent choice to be used among the training data and predictions by the model. Instead of predicting an entire likelihood distribution over y, a prediction of the statistic of y by conditioning on x is made. The loss function applied to train a neural network frequently comes with a regularization term. For deep

neural networks conventionally, a weight decay method is adopted (Goodfellow, Bengio, & Courville, 2016). Usually, weight decay is performed by minimizing a sum comprising both the Mean Squared Error (MSE) on the training and a criterion $L(w)$ that expresses an inclination for the weights to have less squared ℓ_2 norm (Goodfellow et al., 2016). Particularly,

$$L(w) = MSE_{train} + \lambda w^T w \tag{2.1}$$

where in the equation 2.1 λ represents a value that is picked ahead of time that regulates the strength of preference for smaller weights (Goodfellow et al., 2016). When $\lambda = 0$, no preference is imposed, and a more extensive λ makes the weights to become smaller. Minimizing $L(w)$ leads to a selection of weights that make a tradeoff between fitting the training data and being scanty, which produces solutions that have a gentle slope or place weight on fewer of the features.

Maximum likelihood (the negative log-likelihood) is the commonly used method to train most modern neural networks; in other words, it is expressed as the cross-entropy within the model distribution and the training dataset. This equation gives the explained loss function:

$$L(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{data}} \log p_{model}(y \mid x) \tag{2.2}$$

The particular class of $\log p_{model}$ is the foundation of the particular design of the loss function, which varies in different models. Deriving the loss function from maximum likelihood utilizing this approach has an essential advantage. It eliminates the difficulty of outlining loss functions for every model. While designing a neural network, the inclination of the loss function is obligated to be broad and foreseeable adequately to work being a

useful director during the training.

For many models, negative log-likelihood is employed, so several output units comprise an exponential function, which helps to saturate if its argument has a hugely negative value. Then it can be used to cancel the exponential of some output units (Goodfellow et al., 2016). Utmost models are parameterized in such a way that for discrete output variables so they are not able to denote a likelihood of zero or one but still can be approaching doing so arbitrarily. As an instance, logistic regression is such a model. By using this approach, the model can designate a remarkable large density to the actual target, which results in cross-entropy reaching negative infinity (Goodfellow et al., 2016).

Softmax Units for Multinoulli Output Distributions: Besides choosing the loss function, it is crucial to select an output unit. In most cases, cross-entropy among the data and distribution of the model can be the right option. The identical neural network unit can be employed as both output and hidden units. So it is assumed that a provided set of hidden features by the feedforward neural network represented by $h = f(x; \theta)$. Output units then present a transformation of the features to accomplish the job that the network needs to fulfill. Softmax function most likely is used whenever we want to denote a probability distribution across a discrete variable within probable values.

The sigmoid function is also used over a binary variable to express a probability distribution the same as Softmax; in other words, Softmax is a generalized form of the Sigmoid function. In most cases, Softmax functions are used classifier output, to designate the probability distribution across various classes. The model can use Softmax functions inside itself in case we desire the model to pick within one often different choices for some internal variable but is not frequent. For binary variables where it is expected to generate

$$\hat{y} = P(y = 1|x). \tag{2.3}$$

which is a single number, since mentioned number required to be within 0 and 1 and also the result of logarithm function for that number wanted to be adequately reformed toward gradient-based optimization of the log-likelihood, a number $z = \log \tilde{P}(y = 1|x)$ is predicted instead. Sigmoid function controls a Bernoulli distribution of the exponentiated and normalized value. A vector $\hat{y}$, with $\hat{y}_i = P(y = i|x)$ helps to generalize for a discrete variable which has n values. Despite, every element of $\hat{y}_i$ be between 0 and 1, but also the summation of the entire vector items should be one which leads to a valid probability distribution.

The multinoulli distribution uses the same way as the Bernoulli distribution uses to have generalized form. Initially, raw log-likelihoods are calculated using a linear layer:

$$z = W^{\top} h + b \tag{2.4}$$

which in this equation $z_i = \log \tilde{P}(y = i|x)$. Later on, to attain the desired $\hat{y}$, z can be exponentiated and normalized by Softmax function. The Softmax function can be illustrated by the given equation below:

$$softmax(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)} \tag{2.5}$$

Same as logistic sigmoid, the exp function performs very well to train the Softmax in a way so it can generate value in the output units by adopting maximum log-likelihood, which is our desired target value y. Here, main aim is to have maximum value for equation $\log P(y = i; z) = \log softmax(z)_i$. An essential characteristic of the defined terms is that log in the log-likelihood can reverse (it is natural) the exponential function of the Softmax:

$$\log softmax(z)_i = z_i - \log \sum_j exp(z_j) \tag{2.6}$$

In equation 2.6, the first term has a constant straight contribution to the loss function, and because it is not able to saturate, the training can continue, despite though the effect of z_i on the following term of equation 2.6 can become very small. The first term encourages z_i to be upshifted while maximizing the log-likelihood. At the same time, the following term helps every z to be downshifted. To have better discernment about the following term, $\log \sum_j exp(z_j)$, perceive that $max_j\ z_j$ approximates aforementioned term. The idea for this approximation is that $exp(z_k)$ is unimportant to consider for any z_k that is noticeably less than $max_j\ z_j$. This approximation gives the intuition that the log-likelihood with negative value loss function constantly well penalizes the most intense prediction that is wrong comparing to the true target. In case that the expected result already possesses the most significant amount to the Softmax, then the $-z_i$ term and the $\log \sum_j exp(z_j) \approx max_j z_j = z_i$ terms will approximately drop. Altogether, the model is controlled by unregularized maximum likelihood to determine those parameters which lead the Softmax to prediction with probability values:

$$softmax(z(x;\theta))_i \approx \frac{\sum_{j=1}^{m} 1_{y^{(j)}=i, x^{(j)}=x}}{\sum_{j=1}^{m} 1_{x^{(j)}=x}} \tag{2.7}$$

The advantage of a consistent estimator like maximum likelihood is that it is guaranteed that the prediction happens as long as the model group can represent the training distribution. Practically, when the model is limited to the approximation of these fractions, it means that the model has limited capacity and imperfect optimization. The exp becomes hugely negative for the argument, and then it leads to a vanishing gradient, so in such case, only those target functions that use a log to reverse the exp of the Softmax can

escape this failure through learning.

Squared error performs weakly as a loss function and usually causes failure during training for Softmax units to change its output. Both sigmoid and Softmax can saturate. When the input to the sigmoid function is much negative or positive, then it will saturate. The output from Softmax is not a single value, so that these values may saturate due to the contrast among input values get severe. So as it is expected when the Softmax saturates, common loss functions that deal with the Softmax also most likely will saturate, except those loss functions can reverse the saturation of activation function. Considering the equation below:

$$softmax(z) = softmax(z + c). \tag{2.8}$$

we can achieve a numerically solid modification inherited from Softmax:

$$softmax(z) = softmax(z - max_i z_i). \tag{2.9}$$

The second equation empowers users to assess Softmax by only little numerical errors, even if z holds notably positive or negative values. The value drives the Softmax function that its parameters deviate from $max_i\ z_i$.

Considering an output $softmax(z)_i$, if the corresponding input has the maximum possible value ($z_i = max_i z_i$) and z_i is remarkably more massive than rest of the inputs it will saturate to 1. Alternatively, it will additionally saturate to 0 if z_i has not the maximum possible value; moreover, the maximum value is considerably high. In the generalized form, sigmoid units saturate; furthermore, if the loss function cannot compensate for it, then it can head to similar barriers while training. Two various methods exist to generate term z in the Softmax function. The commonly recognized way is to ordinarily have

another layer of the NN before outputting each member of z, as outlined before applying $z = W^\top h + b$. It is possible to force a constraint that a single component of z is set to have a fixed value. As an instance, to demand that $z_n = 0$, which is how precisely the sigmoid function behaves. Defining $P(y = 1|x) = \sigma(z)$ is equivalent to defining $P(y = 1|x) = softmax(z)_1$ with a two-dimensional z and $z_1 = 0$. Both the n argument and $n - 1$ argument, then approaches to the Softmax, can express the corresponding combination of probability distributions except with various training dynamics. Practically, there is seldom enormously differ among using the over parameterized or the limited one, and it is simpler to implement the over parameterized one. The Softmax outputs, in any case, will summate to 1, so increasing the amount of single unit unavoidably corresponds to a drop in the amount of other units. By the maximum (if the difference among the maximum possible value a_i and the others are very high in magnitude), it converts to a winner-take-all form (it happens if a single unit of outputs is almost 1, and the rest are approximately 0). The phrase soft in the Softmax function illustrates that it is for continuous values; moreover, it is differentiable too, and arg max function, besides its result, described as a one-hot vector, is not continuous or differentiable. So, in other words, the Softmax function, therefore, implements a softened form of the arg max function. The analogous soft form of the maximum function is $softmax(z)^\top z$.

Hidden Units So far, the discussion is focused on gradient-based optimization that is commonly used to train most parametric machine learning models of NNs. There is a unique issue to feed-forward neural networks that concerns about what kind of hidden units we need to employ. The study of hidden layers is a notably hot field for study and research, and it seems still have not various ultimate theoretical principles. As an example, ReLus are excellent first selections for hidden units. Although several varieties of hidden units exist, it is usually hard in many cases to determine when to use which sort

of hidden units.

Neither of each hidden units covered in this list is differentiable in every input point. In many examples, hidden units takes a vector x, calculating an affine transformation $z = W^\top x + b$ and next z is input for an element-wise nonlinear function $g(z)$. Hidden units are recognizable by realizing the form of the activation function $g(z)$.

ReLu and generalized forms of ReLu Rectified linear units or generally known as ReLus utilize the activation function $g(z) = max\{0, z\}$. In other words, it gives an output of z if z is positive; otherwise, the output will be 0. Due to its similarity to linear units, it is straightforward to optimize those units. Although it seems similar to linear in the positive axis, the Relu activation function is entirely nonlinear, and also any combinations of ReLu are also known as a nonlinear function. The contrast between a linear unit and a ReLu is that a ReLu produces zero for negative values. So whenever the ReLu is active, then the derivatives over a ReLu stay big. The rectifying operation has the second derivative of 0 for almost all possible values, and when the unit is active, the derivative of the rectifying operation is 1 for all related values.

ReLus are usually utilized across an affine transform function:

$$h = g(W^\top x + b) \tag{2.10}$$

To apply good practice to set the initial value of parameters of the affine transformation, that is possible to make all components of b to hold a small positive number, for example, 0.15. It is also required to have ReLus to be initially active, so this small positive number lets the derivatives to pass through for many inputs in training set to make that happen. Various generalized forms of ReLus are practiced among top recent researches. Many of these generalized forms functioning well comparable to ReLus and

in some cases, appear to perform better. Gradient-based methods are not a wise choice to train a model based on ReLus for which their activation is zero, and this is one of the downsides for ReLus. Different generalized forms of ReLus ensure that they obtain gradient throughout. Basics of certain different generalized form of ReLus is to use a non zero slope α_i when $z_i < 0 : h_i = g(z, \alpha)_i = \max(0, z_i) + \alpha_i \min(0, z_i)$. Absolute amount of rectification will fix $\alpha_i = -1$ to gain $g(z) = |z|$. Other generalized forms of ReLus are more predominantly applicable. For example, a Leaky ReLu makes α_i a little amount like 0.01, while a parametric ReLu handles α_i as one learnable parameter.

Maxout units generalize ReLus by dividing z into sets of k values as an alternative way to applying an element-wise function $g(z)$. The maximum element of one of these groups will be output from each Maxout unit:

$$g(z)_i = \max_{j \in G^{(i)}} z_j, \tag{2.11}$$

here for each group $i, \{(i-1)k+1, \ldots, ik\}$, the $G(i)$ will be list of indices into that groups' inputs. This approach implements a method of learning a multi-direction responsive piecewise linear function for input x space.

A Maxout unit has this ability to learn a linear, also convex piecewise function limited to k pieces. In other words, a Maxout units layer merely is a layer where the activation function is the max of the inputs. Maxout units may, therefore, considered as learning the activation function itself. Generally, it has such a learning power, which makes it able to approximate any convex function as long as the k is big enough.

To be more precise, only two pieces, a Maxout layer, implements the identical function of the input x as a conventional layer by ReLu, Leaky ReLu, PReLu activation function, or a different function altogether. However, the Maxout layer is parameterized

differently from the mentioned layers. As a result, it has distinctive learning dynamics in all aspects in any case. For each Maxout unit, there are k weight vectors rather than only one; as a result, regularization is a more crucial requirement for Maxout units comparing to ReLus. However, if the training set is big enough and at the same time, the amount of pieces per unit is maintained low, they can act appropriately even without regularization.

In some cases, Maxout units can be requiring fewer parameters, which makes it be statistically and computationally even more beneficial. To be more specific, if we assume the features obtained through n separate linear filters, the Maxout units will reduce features k times without dropping information by taking the maximum value over each group of k features. ReLus and all derived generalized forms are built against the idea that models are more straightforward for optimization if their performance is resembling linear models, which applies in various contexts besides deep linear networks.

Sigmoid or Logistic function is a mathematical function possessing an "S" formed curve (sigmoid curve). Frequently, sigmoid function refers to the particular case of the logistic function shown in figure 2.1 and defined by the equation 2.12:

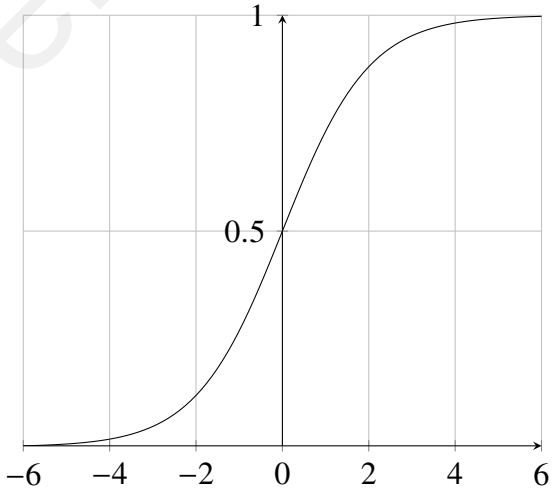

Figure 2.1: Sigmoid or Logistic function

$$S(t) = \frac{1}{1 + e^{-t}} \tag{2.12}$$

Its Range is between 0 and 1. It is straightforward to comprehend and to use, but it possesses essential causes to make it fall out of popularity, and there are reasons for that:

- Sigmoids usually converge very slowly.
- Sigmoids saturate and destroy gradients.
- Its output is not zero centered, which causes the new gradient values to go remarkably far in various directions. 0 < output < 1, and it makes optimization harder.
- Small or vanishing gradient problem which network starts to refuse to learn.

2.1.1 Autoencoders

Autoencoders are conventional neural networks to reduce feature dimensions. In this paper (Y. Wang, Yao, & Zhao, 2016), research has been established that compares auto-encoder and its ability to reduce the dimensionality with the other well-known methods such as LDA, PCA, and Isomap. Their experiments (from (Y. Wang et al., 2016)) show that auto-encoders act differently, and besides, to reduce dimensionality, it can detect repetitive structures as well. So auto-encoder is the right solution for data with repetitive structures (Y. Wang et al., 2016). As another example in paper (W. Wang, Huang, Wang, & Wang, 2014), a generalized autoencoder has been introduced, which is a neural network framework for dimensionality reduction. The mentioned approach in (W. Wang et al., 2014) is compared with conventional approaches like LDA, PCA, LLE, LE, MFA, and Isomap, which shows that autoencoders are performing very good in many possible applications.

Autoencoders are a class of neural networks that reasonably learns to try to make a very close but not exact copy of its input through one or more hidden layers to its output

layer. As it is mentioned, it usually has deep or shallow layers of hidden h units that are also called the encoded layer(s) of an autoencoder. It has to main parts, which are encoder and decoder. Encoder is known as function $h = f(x)$ and decoder which reconstruct input gain is known as $r = g(h)$. The design is shown in Figure 2.2. If an autoencoder successfully learns in the training process to set $g(f(x)) = x$ for all input x, then it is not helpful. Preferably, autoencoders are intended to be incapable of learning to replicate perfectly to do so autoencoders are usually limited with policies that allow them to generate just approximation of data and to choose only input that can resemble the training data as network output. Autoencoders are designed only to learn the useful properties of the data, so the model is designed in a way to determine which features of the input should be selected based on their priority. The new generation of autoencoders are based on stochastic mappings and it no longer limited to deterministic functions $P_{encoder}(h \mid x)$ and $P_{decoder}(x \mid h)$. (Goodfellow et al., 2016).

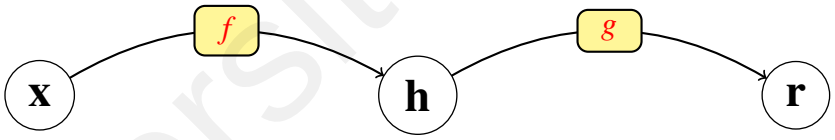

Figure 2.2: Almost all autoencoder types have a typical design as above which includes two main components: encoder $h = f(x)$ and decoder $r = g(h)$. Former maps input to an internal representation h, latter maps representation h to reconstruction r.

Undercomplete Autoencoders: Although reproducing the input in the output layer may seem worthless, but the output of the decoder is not the aim to have this network. Instead, the primary goal is to train the autoencoder to do the input replication part, which will lead to h obtaining useful features. When the autoencoder is forced to have a hidden layer of h with a smaller dimension than the input layer x, it can gain useful features. An autoencoder that is using this method is called under complete. So the autoencoder has to obtain the most notable features of the training data. To train an autoencoder, we need

a process which it is explained clearly as minimizing a loss function:

$$L(x, g(f(x))) \tag{2.13}$$

where L is a loss function, in this loss function, every time x is different from the reconstructed value in the output layer, $g(f(x))$ will be penalized. In such a case, mean squared error can be employed as the loss function. When a decoding layer is linear, and L is the mean squared error, an under complete PCA and a single layer autoencoder will resemble almost equivalent functions if only the autoencoder has a linear transfer function; however, they cannot reproduce same encoded values. (Goodfellow et al., 2016).

Moreover, autoencoders have a nonlinear encoding function f and nonlinear decoding function g can, therefore, learn a robust nonlinear generalized form of PCA. Unfortunately, one of the downsides is that it can appear not to be able to extract useful features if either encoder or decoder are given exceedingly much capacity.

2.1.2 Convolutional Neural Networks (CNN)

CNNs, are a class of ANNs which is suitable to process data that is presented in the multi-dimensions matrix. CNNs have shown remarkably successful performance in useful applications. The term convolutional neural network designates that the network uses convolution, which is a mathematical operation. Convolution is a particular class of linear operations. CNNs are ANNs that utilize convolution operation for matrix multiplication in feature learning or extraction layers.

Convolution: Convolution is a mathematical operation that applying them on functions f and g will generate a different function which is comparable to cross-correlation. It has many use-cases such as probability, pattern recognition, natural language processing, image processing. The convolution operation over function f and function g is shown as

$f * g$ for continues values (Press, 1989).

Because in computer vision, we are dealing with images which are $2D$ data, so as it is explained in (Damelin & Miller Jr, 2012), we can extend convolution from (Press, 1989) to be used for $2D$ discrete values. Here is the convolution operation function for $2D$ images:

$$(f * g)[m, n] = \sum_{u=-U}^{U} \sum_{v=-V}^{V} f[m-u, n-v]g[u, v] \tag{2.14}$$

Convolutional layers: Neural networks for image classification are made up of layers, and each layer can contain hundreds, even thousands of neurons. The earlier layers of the NN closer to the input will focus on the more granular details of the image, such as pixels. Later layers will focus on abstractions. Images contain very complex dense information, and when we work with them using neural networks, we are likely to have a parameter explosion. As an instance, if we have 100 by 100 image, It is possible to set up the first layer to have 10000 neurons to process the individual pixels. If we connect these neurons in the next layer, even if they are fewer than 10000 neurons in the subsequent layers, it will soon run into a case where there are millions of parameters to train within the neural network, which is not sustainable. Therefore it does not make that much sense to work with images employing a fully connected NN. In such a case, CNNs come in, enabling NNs to get these parameters under control.

One significant advantage of CNNs is that there are dramatically fewer parameters than deep neural networks for similar performance. CNNs work particularly well with data that come expressed in two dimensions, such as an image. It has a height and a width and pixel values, which make up the individual elements of the matrix. CNNs use convolution to have neurons focus on local receptive fields on smaller portions of the input image instead of the whole image. The primary role of convolutional layers in CNNs is to

zoom in on specific bits of input. Imagine that convolutional layers are a pair of binoculars, which is used to look at small bits of the input at a time and have higher layers piece these together. Convolutional layers are followed by higher-level layers that aggregate inputs into more abstract or higher-level features. CNNs might include convolutional layers which focus on pixels which are aggregated to lines which are then aggregated to contours or edges, and finally abstracted to identify objects. So far, we know one thing about convolutional layers on CNNs, which is the fact that they convolution. We assume that we have an input image in which this image is made up of pixels. Imagine now a matrix of neurons but not a layer of neurons, which are receptive to the convolutional kernel sliding over this pixel matrix. This matrix of neurons that respond to a convolutional kernel is called a feature-map. A convolutional layer is made up of a number of these feature-maps where each feature-map responds to a different convolutional kernel. The convolutional kernel that we slide over the input matrix forms the local receptive field for every neuron in this feature-map. The size of this kernel or filter, as it is also called, determines how many elements of the input matrix will be processed (by a particular neuron). In the first layer, this input matrix will comprise of the pixel values of the image. In subsequent layers, the input matrix will comprise outputs of neurons from the previous layers. This convolutional kernel is expressed regarding the width and height of the receptive area, how many neurons in its width, and how many neurons in its height where a neuron represents a matrix element. The convolutional kernel is the zoom-in mechanism for identifying image characteristics, so using small convolutional kernels tends to be more efficient, so it is better to use two 3×3 kernels, as compared with one 9×9 kernel. Stride determines the distance between successive receptive fields. The stride determines how the convolutional kernel slides over the input. As the kernel slides over, a different neuron is activated through that receptive field. Zero paddings determine how is a desirable way

to treat the pixels at the edges of the image. Feature-map contains neurons that have identical weights and biases, which are determined by the weights that are specified in the kernel, which acts as the feature detector for the feature-map.

The weights of the kernel, and thus, of the feature-map are determined during the training process. There are two significant advantages to having feature-maps with neurons that respond to local receptive fields that are dramatically fewer parameters to train, mainly because all neurons in a feature-map have the same weights and biases. Feature-maps also allow convolutional neural networks to recognize patterns independent of the location. If there is a pattern in the top left corner of the image, so if there is a cat in the input image, it will be identified no matter where it is located. It is essential to realize that these feature-maps, which we stacked together, make up convolutional layers, the neurons are not connected to all pixels individually. A group of pixels or a group of neurons are connected to one neuron in the next layer. Thus, convolutional neural networks are set to be sparse neural networks. One convolutional layer in a CNN comprises of several feature-maps all of the equal sizes. Each of these feature-maps responds to different convolutional kernels that have slid over the input, which means each of these feature-maps has different parameters. Convolutional neural networks will have some convolutional layers stacked up one after another. They may be interspersed with pooling layers. Every neuron within a feature-map within a convolutional layer includes the feature-maps of all previous layers. A neuron is in a particular feature-map in one convolutional layer can see all previous feature-maps in its local receptive field, and this is the secret of how aggregated features are picked up using convolutional neural networks. CNNs can pick up higher-level aggregations by having neurons respond to multiple feature-maps within their local receptive field. Pooling layers also play an essential role here.

Pooling: The second kind of layer that a convolutional neural network is made up of is the pooling layer (Figure 2.3). Neurons in a pooling layer have no weights or biases, so they do not have parameters that need to be trained during the training phase of the neural network. A pooling neuron applies merely some aggregation function to its input. The pooling layer performs a sub-sampling of the input to extract the most significant features. Common aggregations operations performed by the pooling layer are maximum, sum, or average of the inputs. Pooling is some sliding window function, but within each window (for example, a 2×2 window), you will perform a simple aggregation over all cells that are included. The maximum value (for max-pooling) of all the input cells within the 2×2 window will be pooling results. The window will move in the whole input matrix, and output will be a sub-sample of this matrix. The same procedure can be applied in case of average or sum pooling aggregation function. Subsampling of inputs for such a dense problem such as image recognition allows us to reduce memory usage while training the neural network substantially. Pooling also will enable us to mitigate the issue of overfitting our model on the training dataset. Pooling also allows the neural network to recognize features independent of the location, which is called location invariance. Pooling layers typically perform aggregations on each channel of the input image if they are colored images or each feature-map of the previous layer independently. At the same time, pooling retains an only corresponding portion of the input that is passed into it, so the output area of a pooling matrix is less than the area that was input for the pooling matrix, so the image becomes smaller and smaller as it passes through pooling layers. However, the depth of the image as represented by the feature-maps in the convolutional layers remain the same when they pass through a pooling layer. Pooling layers change the size of an image, but not the depth. Then we have fully connected layers that consist of a feed-forward neural network connected to the end of typical convolutional neural

networks that usually have few fully connected layers that typically have ReLU activation functions for their neurons. To classify the input image, the neural network has to assign the output to enable that image finally. A common choice for this layer is the Softmax layer that we have discussed earlier.

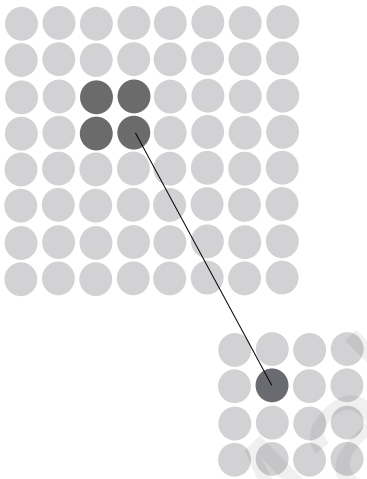

Figure 2.3: Max pooling

2.2 Facial Feature Extraction

Automatic facial feature extraction is one of the most fundamental problems in computer vision, which is a necessary step in face recognition, facial image compression, facial age, and facial gender classification. Many approaches are proposed to solve this problem. All of these methods for facial feature extraction are divided into two main categories:

1. Supervised Feature Extraction Techniques

2. Unsupervised Feature Extraction Techniques

2.3 Supervised Feature Extraction Techniques

Supervised feature extraction is using a supervised feature learning method to learn features of labeled data. Labeled data here are the data with manual features that human marks those features as target features or labels. A model that is trained by these labels will

be employed to predict similar features. These labeled data allow the model to compute an error term then be used as feedback to correct the learning process (for example, a gradient descent backpropagation neural network).

2.3.1 Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) is one of the robust feature extraction methods in image processing and computer vision (Dalal & Triggs, 2005; Felzenszwalb, McAllester, & Ramanan, 2008). It has been used to detect human or pedestrian (Zhu, Yeh, Cheng, & Avidan, 2006), to classify age and gender (Verma & Jariwala, 2018) or to recognize facial expression (Nazir, Jan, & Sajjad, 2018; Nigam, Singh, & Misra, 2018).

The idea is to count the number of gradient orientation occurrences in specific parts of an image. Initially, the image is partitioned into 8×8 regions without overlapping pixels or cells. Then a 1D HOG will be accumulated over pixels for each cell. Two essential components of these histograms are 1) they capture local shape features, but 2) they are invariant to slight deformations. HOG process discretizes the gradient into one of nine orientation bins toward each pixel base. The orientation is determined based on votes provided by the respected pixel. The gradient magnitude at that pixel specifies the strength of that orientation. Concerning the computation of the gradient for colorful images, each pixel at each color channel will be processed independently, and the most significant gradient magnitude among those channels will be picked. Lastly, based on the gradient strength in a region near each cell, the histogram of that cell is normalized. The process is done over four 2×2 blocks of cells. Each block contains a particular cell, and normalizing the given cell histogram is done by considering the total energy in each block. As a result, there will be a 9×4 vector, which represents the regional gradient information for each cell. HOG features for each level of a standard image pyramid will form the final step, which is to provide a HOG feature pyramid (see Figure 2.4). At the bottom of the

pyramid, there are more detailed gradients histogrammed features comparing to features at the top of the pyramid, which are gradients histogram computed over fairly broad areas (Felzenszwalb et al., 2008).

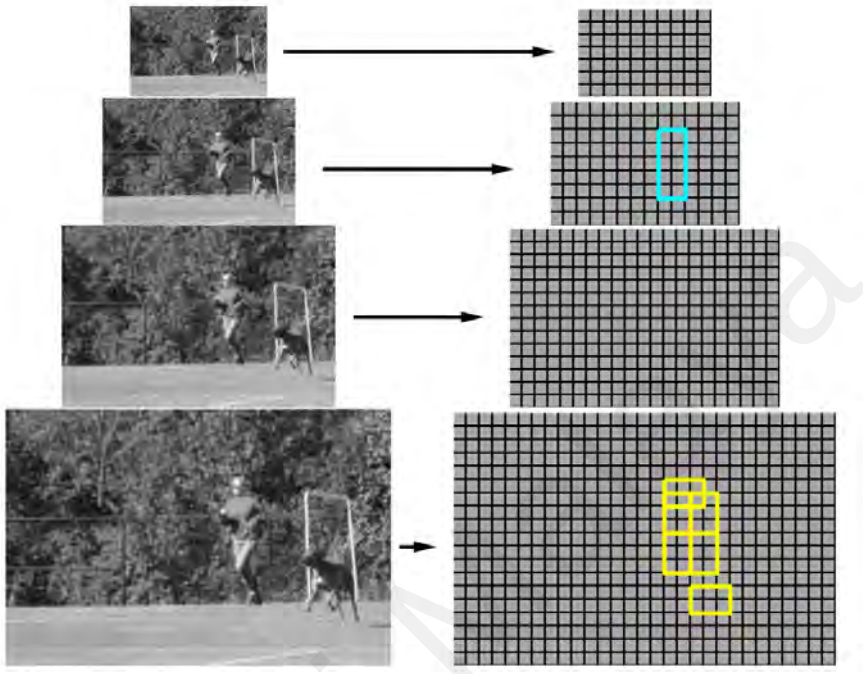

Figure 2.4: HOG features pyramid

2.3.2 Action Units

Facial Action Coding System or FACS is a coding system that has this ability to code almost all possible facial expressions manually. It deconstructs it into the particular Action Units (AU) that produce the facial expression. AUs are suitable to be adopted for any higher-order decision-making process since AUs are independent of any interpretation, which includes essential emotions recognition, or commands for intelligent environments based on human facial expression.

In (Baltrušaitis, Mahmoud, & Robinson, 2015), they have proposed a robust technique to obtain these action units. They are dependant on two main kinds of features: geometry features and appearance features. For obtaining these features, it is required to track specific landmarks and face alignment on a facial image, which is explained in detail

in the paper by (Baltrušaitis et al., 2015).

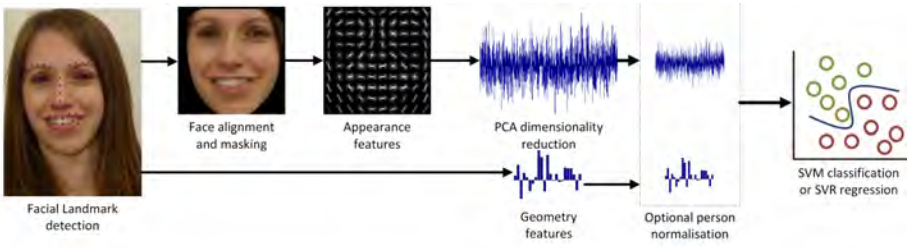

Figure 2.5: Action Units detection pipeline overview.

2.4 Deep Feature Extraction Techniques

Deep feature extraction methods are used in different aspects. For instance, in (Y. Chen, Jiang, Li, Jia, & Ghamisi, 2016), to classify hyperspectral image (HSI), they have introduced a regularized deep feature extraction (FE) technique which relies on a convolutional neural network (CNN). Their method applies different convolutional and pooling layers to obtain unsupervised, nonlinear, discriminant, and invariant features from HSIs. Features, as mentioned above, are helpful regarding image classification. Common issues for the classification of HSI are 1) imbalance between high dimensionality and 2) limited availability of training samples, which are solved by a few strategies such as ℓ_2 regularization and dropout. These strategies help to escape from overfitting in class data modeling.

In (Zhao & Du, 2016), they have proposed a spectral-spatial feature-based classification (SSFC) structure that uses dimension reduction and DL methods for spatial and spectral feature learning. Regarding high-dimensional hyperspectral data sets, their structure introduces an algorithm that can extract spectral features out of such a dataset. CNN is employed to obtain spatial-related features. Stacked spectral and spatial features together will extract the fusion feature to classify by the multiple-feature-based classifier as image classification.

In (Sun, Chen, Wang, & Tang, 2014), they extracted useful features using CNN or

ConvNets. Two crucial part of ConvNets is namely convolution and pooling, which are used to extract visual features in a hierarchy form. Concerning $2D$ images as input, they can be from small patches to more abstract concepts like Cat or Human faces. They have used four convolutional layers, with locally shared weights for the last two convolutional layers. It obtains 160-dimensional DeepID2 features at the DeepID2 layer. The DeepID2 layer is followed by the fully-connected layer, which is connected to both the last two convolutional layers. They have used ReLu units in their entire network. This network is used to extract features from RGB input with dimensions of 55×47.

2.5 Extreme Learning Machine

The extreme learning machine or ELM has become an essential problem-solving approach for the past few years. There are many important research topics using ELM for machine learning due to its novel properties, such as notably fast training, robust generalization, and predicting or classification strength.

2.5.1 Learning in Extreme Learning Machine

Concerning single hidden layer feedforward networks or SLFNs, ELM is a robust solution for them, which is proved to have excelling training accuracy and speed in numerous applications, for instance, face classification. One prominent advantage of the ELM networks is that their parameters of hidden units are randomly generated, and they do not need to be fine-tuned, on the other hand, conventional learning algorithms such as backpropagation, support vector machine, they usually need a kind of iterative training algorithm to tune the hidden units. In theory, a single layer feedforward neural network that has hidden neurons that are created randomly following by output weights which are fine-tuned using regularized least square, it can still keep its general prediction ability without the need to adjust the parameters of hidden units. So ELM manages to deliver

faster and more reliable generalization performance comparing to other iterative models such as SVM or BP-NNs.

Many researchers have been contributed remarkable researches and studies to ELM concerning theory and also applications so far. The capacity of ELM has been empowered with kernel learning, which, as a result, makes ELM a powerful approach for an extended type of feature mapping besides traditional usages. Besides original ELM, which mainly focuses on classification, extended ELM is widely practiced for supervised and semi-supervised networks with magnificent results. The original ELM and its variants mainly focus on classification. The primary attribute of ELM is that a non-iterative linear function can be used for weight, and the reason behind that is there is no dependency between input and output wights like what exists in, for example, BP training algorithms. This attribute leads to a notable improvement regarding training speed in ELM comparing to, for example, Support Vector Machines (SVM) or compared to Multilayer Perceptron (MLP), based on experiments done by (Akusok, Björk, Miche, & Lendasse, 2015).

The Theory of ELM: Assume that the following equation from (Tang, Deng, & Huang, 2016) can represent SLFNs with L hidden nodes:

$$f_L(X) = \sum_{i=1}^{L} G_i(x, a_i, b_i) \cdot \beta_i,\ a_i \in R^d,\ b_i, \beta_i \in \mathbf{R} \tag{2.15}$$

where $G_i(\cdot)$ denotes the i_{th} hidden node activation function, a_i is the input weight vector connecting the input layer to the i_{th} hidden layer, b_i is the bias weight of the i_{th} hidden layer, and β_i is the output weight (Tang et al., 2016). For additive nodes with activation function g, G_i is defined as follows:

$$G_i(x, a_i, b_i) = g(a_i \cdot x + b_i) \tag{2.16}$$

and for radial basis function (RBF) nodes with activation function g, G_i is defined as

$$G_i(x, a_i, b_i) = g(b_i||x - a_i||) \tag{2.17}$$

It has been proved that the SLFNs are able to approximate any continuous target functions over any compact subset $x \in \mathbf{R}^d$ with above random initialized adaptive or RBF nodes. Let $L^2(x)$ be a space of functions f on a compact subset x in the d-dimensional Euclidean space $\mathbf{R}^d$ such that $|f|^2$ is integrable, that is, $\int_x |f(x)|^2 d_x < \infty$ (Tang et al., 2016). For $u, v \in L^2(x)$, the inner product (u, v) is defined by

$$(u, v) = \int_x u(x)v(x)d_x \tag{2.18}$$

The norm in $L^2(x)$ space is denoted as $||\cdot||$, and the closeness between network function f n and the target function f is measured by the $L^2(x)$ distance (Tang et al., 2016)

$$||f_L - f|| = \left(\int_x |f_n(x) - f(x)|^2 d_x \right)^2 \tag{2.19}$$

Theorem 2.1: Given any bounded non-constant piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$, if span $\{G(a, b, x) : (a, b) \in \mathbf{R}^d \times R\}$ is dense in L^2 , for any target function f and any function sequence $g_L(x) = G(a_L, b_L, x)$ randomly generated based on any continuous sampling distribution, $\lim_{n\rightarrow\infty} ||f - f_n|| = 0$ holds with probability one if the output weights β_i are determined by ordinary least square to minimize $||f(x) - \sum_{i=1}^{L} \beta_i g_i(x)||$ (Tang et al., 2016).

Randomly generated networks with the outputs being solved by least mean square can maintain the universal approximation capability, if and only if the activation function g is non-constant piecewise and span $\{G(a, b, x) : (a, b) \in \mathbf{R}^d \times R\}$ is dense in L^2. Based

on this theorem, ELM can be established for fast learning, which will be described in detail in further(Tang et al., 2016).

ELM Learning Algorithm: According to Theorem 2.1, the ELM can be built with randomly initialized hidden nodes. Given a training set $\{(x_i, t_i)|x_i \in \mathbf{R}^d, t_i \in \mathbf{R}^m, i = 1, ..., N\}$, where x_i is the training data vector, t_i represents the target of each sample, and L denotes the number of hidden nodes. From the learning point of view, unlike traditional learning algorithms, ELM theory aims to reach the smallest training error but also the smallest norm of output weights.

$$Minimize : ||\beta||_u^{\sigma_1} + \lambda||\mathbf{H}\beta - T||_v^{\sigma_2} \tag{2.20}$$

where $\sigma_1 > 0, \sigma_2 > 0, u, v = 0, (1/2), 1, 2, ..., +\infty$, $\mathbf{H}$ is the hidden layer output matrix (randomized matrix)

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(x_1) \\ \vdots \\ \mathbf{h}(x_N) \end{bmatrix} = \begin{bmatrix} \mathbf{h_1}(x_1) & ... & \mathbf{h_L}(x_1) \\ \vdots & \vdots & \vdots \\ \mathbf{h_1}(x_N) & ... & \mathbf{h_L}(x_N) \end{bmatrix} \tag{2.21}$$

and $\mathbf{T}$ is the training data target matrix

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix} = \begin{bmatrix} \mathbf{t}_{11} & ... & \mathbf{t}_{1m} \\ \vdots & \vdots & \vdots \\ \mathbf{t}_{N1} & ... & \mathbf{t}_{Nm} \end{bmatrix} \tag{2.22}$$

The ELM training algorithm can be summarized as follows:

- Randomly assign the hidden node parameters, e.g., the input weights a_i and biases b_i for additive hidden nodes, $i = 1, ..., L$.

- Calculate the hidden layer output matrix $\mathbf{H}$.
- Obtain the output weight vector

$$\beta = \mathbf{H}^{\dagger}\mathbf{T} \tag{2.23}$$

where $\mathbf{T} = [\mathbf{t}_1, ..., \mathbf{t}_N]^T$, $H^{\dagger}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$.

The orthogonal projection method can be efficiently used for the calculation of MP inverse: $\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$, if $\mathbf{H}^T\mathbf{H}$ is nonsingular; or $\mathbf{H}^{\dagger} = \mathbf{H}^T(\mathbf{H}^T\mathbf{H})^{-1}$, if $\mathbf{H}\mathbf{H}^T$ is nonsingular. According to the ridge regression theory, it was suggested that a positive value $(1/\lambda)$ is added to the diagonal of $\mathbf{H}^T\mathbf{H}$ or $\mathbf{H}\mathbf{H}^T$ in the calculation of the output weights β. By doing so, the resultant solution is equivalent to the ELM optimization solution with $\sigma_1 = \sigma_2 = u = v = 2$, which is more stable and has better generalization performance. That is, in order to improve the stability of ELM, we can have

$$\beta = \mathbf{H}^T\left(\frac{1}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{2.24}$$

and the corresponding output function of ELM is

$$f(x) = h(x)\beta = h(x)\mathbf{H}^{\mathbf{T}}\left(\frac{1}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{2.25}$$

or we can have

$$\beta = \left(\frac{1}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{2.26}$$

and the corresponding output function of ELM is

$$f(x) = h(x)\beta = h(x)\left(\frac{1}{\lambda} + \mathbf{H}\mathbf{H}^{\mathbf{T}}\right)^{-1}\mathbf{H}^{\mathbf{T}}\mathbf{T} \tag{2.27}$$

2.5.2 High Performance Extreme Learning Machine

The HP-ELM or High-Performance Extreme Learning Machine is introduced by (Akusok et al., 2015) for the first time, which is a powerful toolbox that has implemented ELMs networks precisely. Generally speaking, an ELM is a straightforward algorithm, which is only a few lines of code in Matlab. However, the performance of such ELMs is not optimal. It is essential to manage to do regularization, parameter selection efficiently, to be flexible for scalability without facing out-of-memory issues to process on Big Data, which leads to achieving the best accuracy. HP-ELM toolbox has provided the best performing ELM implementation for researchers to focus on researches without concerning about the implementation of these networks from scratch and also to be more flexible to deploy the research to production and commercial products.

2.5.3 Hierarchical Extreme Learning Machine

Feature extraction is an essential requirement for classification. In a hierarchical(multi-layer) learning design using ELM autoencoders, the initial inputs are fed into various hidden layers, and then the outputs of the previous layer are used as the inputs of the next one.

Autoencoders based on ELM Theory: ELM method can be applied to form an autoencoder for an MLP, which then the autoencoder can be used as a feature extractor. As it is explained in autoencoder section 2.5.1, It utilizes the encoded outputs to approximate the original data input. The critical difference between normal autoencoder and ELM based autoencoders is that ELM uses randomly mapped outputs. So, the input reconstruction can be seen as an ELM training problem, although it needs typically regularized least mean square optimization for non-elm approaches. Features that the ELM autoencoder extract tends to be dense and may have redundancy, and the reason is that original ELM

uses a sort of penalty, which in such case, we need a sparse autoencoder instead. In further, we will explain ELM Sparse Autoencoder, which is employed as the essential element of H-ELM.

Hierarchical-ELM is made in a hierarchical structure, and the design is different from conventional layerwise learning in deep learning frameworks. H-ELM is made of two primary and essential stages. First, unsupervised feature learning and extraction, second, supervised encoded feature classifying. Concerning the earlier stage, a sparse ELM autoencoder is utilized to obtain sparse features in various levels from the input, while for the later stage, the original ELM is employed for classifying. In further, we will explain more details about H-ELM and also its advantage from different perspectives. As a preprocessing step, the raw input needs to be converted into an ELM random feature space. Afterward, the preprocessed input will go through an N-Layer unsupervised sparse feature learning hierarchy. The output from every hidden layer is described in this equation:

$$H_i = g(H_{i-1} \cdot \beta) \tag{2.28}$$

here ($i \in [1, K]$) and it is the index of the hidden layer. The hidden layer H_i and H_{i-1} are output from respectively i and $i-1$ hidden layer. The output weights are shown as β and $g(\cdot)$ is the activation function for hidden units.

Each hidden layer performs as an independent feature extractor unit. In each layer, computed features get more compact. Moreover, the wights of each hidden layer will be set only one time and without the need for fine-tuning when features from that layer are extracted, which is an entirely different story comparing to backpropagation based neural networks, which need to be fine-tuned iteratively. As a result, H-ELM is quicker than deep neural networks concerning the training speed (Tang et al., 2016).

The output from the very last hidden layer of sparse autoencoder H_K is prepared for

the classification stage, which is the high-level compact features that hierarchical layers have extracted those from input layer feeds. They are perturbed by random before they go through the classification stage, and finally, they are used as the inputs to the supervised classifying stage to produce final outputs of the H-ELM model.

As it is explained in (Tang et al., 2016), H-ELM is based on random feature mapping, which fully utilizes the approximation ability of ELM networks regarding feature learning and classification. Based on Theorem 2.1 from (Tang et al., 2016), H-ELM can approximate or classify any input data.

Sparse Autoencoder for H-ELM: H-ELM owns two independent stages, unsupervised and supervised training. In this section, we will concentrate on how to train the sparse autoencoder of the H-ELM design. As we explained earlier, autoencoder intends to learn a function $h_\theta(x) \simeq x$, where $= \{A, b\}$, A are the hidden weights, and b is the bias. In plain English, the autoencoder attempts to approximate a function that reconstructs similar values as the input into the outputs (Tang et al., 2016).

ELM sparse autoencoder, unlike the autoencoders practiced in the conventional deep learning algorithms, the input weights of the ELM sparse autoencoder are trained by exploring the path back of a random space. The ELM theory has proved that the training of ELM with randomly mapped input weights is capable of approximating any input data. So if the same approach used to train the autoencoder, as soon as the autoencoder is initialized, we no longer need to fine-tune it. Besides sparse autoencoder is using ℓ_1 optimization which leads to a more compact and sparse features to be extracted by the autoencoder

ELM sparse autoencoder optimization model is expressed as below:

$$O_\beta = \underset{\beta}{argmin}\{||\mathbf{H}\beta - \mathbf{X}||^2 + ||\beta||_{\ell_1}\} \tag{2.29}$$

where here X is the input value, H represents the random mapping output, and β is the hidden layer weight. In the conventional deep learning algorithms, typically X is the encoding outputs of the bases β. These parameters will be adjusted during the iterations of optimization. Despite the case that in sparse autoencoder, since random mapping is employed for hidden units, in this case, X is the data, and H is the random initialized output, which then optimization is not required.

Moreover, the experiments that are done by (Tang et al., 2016) has proved that it has improved both training time plus the training accuracy. The optimization algorithm for the ℓ_1 optimization problem in detail can be found here (Tang et al., 2016), but we will discuss a bit about essential parts. Here is the object function:

$$O_\beta = p(\beta) + q(\beta) \tag{2.30}$$

where $p(\beta) = ||\mathbf{H}\beta - \mathbf{X}||^2$, and $q(\beta) = ||\beta||_{\ell_1}$ is the ℓ_1 penalty term of the training model.

In this step, a FISTA algorithm (Beck & Teboulle, 2009) is employed to minimize a smooth convex function with a complexity of $O(1/j^2)$, where j denotes the iteration times(Tang et al., 2016).

Here are the implementation steps in details:

1) Calculate the Lipschitz constant γ of the gradient of smooth convex function ∇p.

2) Begin the iteration by taking $y_1 = \beta_0 \in R^n$, $t_1 = 1$ as the initial points. Then, for $j(j \geq 1)$ the following holds.

   a) $\beta_j = s_\gamma(y_j)$, where s_γ is given by

   $$s_\gamma = \underset{\beta}{argmin}\left\{\frac{\gamma}{2}||\beta - (\beta_{j-1} - \frac{1}{\gamma}\nabla p(\beta_{(j-1)}))||^2 + q(\beta)\right\}.$$

   b) $t_{j+1} = \frac{1+\sqrt{1+4t_j^2}}{2}$

c) $y_{j+1} = \beta_j + \left(\frac{t_j - 1}{t_{j+1}}\right)(\beta_j - \beta_{j-1})$

To correctly retrieve the data from the corrupted ones, we need to compute the iterative steps above. To recover compact descriptions of the initial data, initially, we consider β being the weights of the sparse autoencoder, and then we can compute the inner product of the inputs and extracted features for that.

ELM vs H-ELM: Training in H-ELM is much better than ELM concerning performance, and as mentioned earlier, the H-ELM has two essential parts. First, unsupervised feature extraction, and second, supervised feature classifying. A very critical difference among H-ELM and ELM is that before feature classification happens, hierarchical sparse autoencoders of H-ELM will extract useful features of the raw input data, while in classic ELM design, the raw data is fed to the network for classification. As a best practice in the data analysis field, the compact features appear to be more efficient in eliminating redundancy of the raw input data, which therefore enhances the overall training performance significantly.

CHAPTER 3: METHODOLOGY

3.1 Introduction

In previous chapters, we discussed conventional solutions for age and gender classification, and we also discussed the limitations we have. In this chapter, we will focus on methods to overcome those limitations were explained in the first chapter. The methodology that is adopted in this research is to propose an architecture to classify age and gender. An age and gender classification system is proposed, which in short it is a machine learning model that extracts handcrafted features, namely HOG and Action Unit combined with unsupervised features using a pre-trained convolutional neural network from facial images then reduced in dimensions by autoencoders and finally classified by H-ELM as the classifier. In this chapter, the architecture that is designed for our research will be described in detail.

3.2 Model Overview

Our proposed model architecture for age and gender classification has essential layers which are:

- Preprocessing
- Initial layer or input layer
- Feature extraction layer
- Feature combining and dimensionality reduction layer
- Classification layer

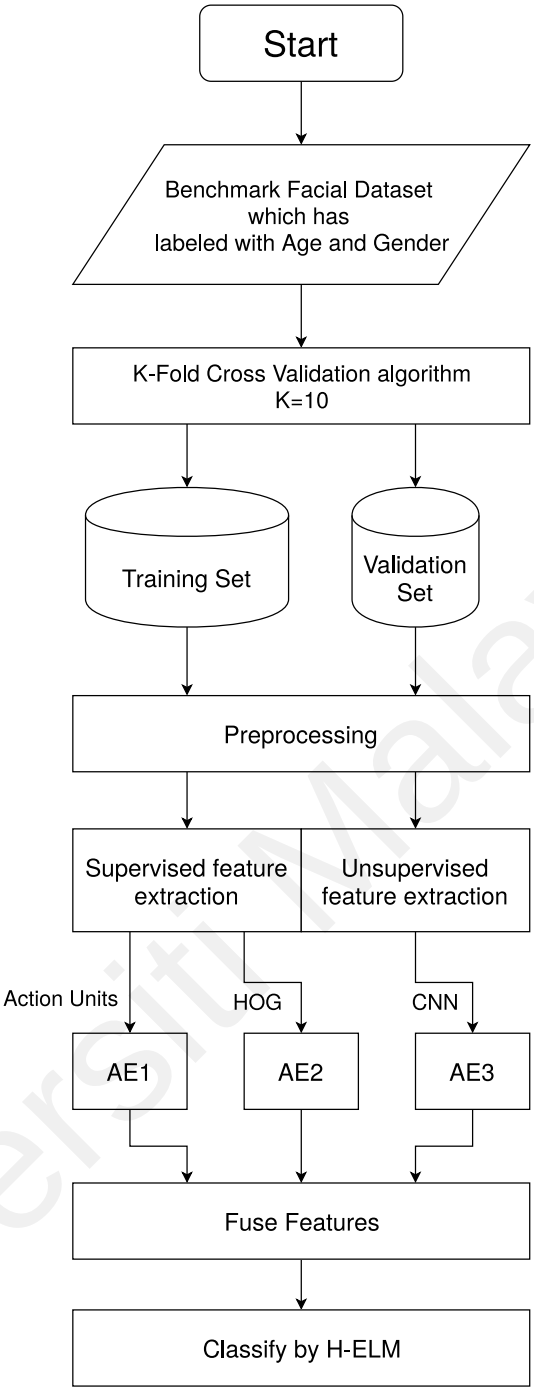

Figure 3.1: Overview of the proposed architecture

In Figure 3.1, we have illustrated an overview of our proposed system. A benchmark dataset is split to training and validation set using a k-fold cross-validation algorithm, then both are sent to preprocessing step, which for example detects face position in input images, normalizes the input data, and finds facial landmarks using the method from (Zadeh, Chong Lim, Baltrusaitis, & Morency, 2017). Then the preprocessed data will

be sent to feature extraction layers where Action Units, HOG features, and CNN features will be extracted simultaneously. Each feature group is sent to the respected autoencoder (AE1, AE2, or AE3) to encode each feature vector to a similar useful feature vector with different dimensions. Outputs from autoencoders are then combined, and a new feature vector is created, which is the output for the H-ELM classifier.

3.3 Preprocessing

Facial images in age and gender labeled benchmark datasets are in different sizes, and also coordinates of faces are different in each picture, and in many cases, there is a background environment that usually causes noise. As the first step, we detect face area in all pictures, and we drop images without a human face or those can not be adequately detected. The face area will be cropped out of the original image, and then it will be resized to a fixed width and height, which are respectively 227 and 227. Since the facial image is an RGB image, the output image has three dimensions, which create an input vector of size $227 \times 227 \times 3$.

3.4 Initial Layer or Input Layer

The initial layer is the input layer to the whole model that is the RGB facial image from the preprocessing step. This image is a matrix that has dimensionality of $227 \times 227 \times 3$. This layer will be fed to the next and first essential layer of our architecture.

3.5 Feature Extraction Layer

In this layer, the essential features of facial images are extracted. In this research, we want to analyze a hybrid feature learning model. We have designed a hybrid handcrafted and deep features extraction model to extract the most useful features. Generally speaking, we have two categories for feature learning which are namely supervised and unsupervised feature learning methods. Earlier in chapter two, we have explained how Action Units (or

AUs) and Histogram of Oriented Gradients (or HOG) are working. AUs and HOG are examples of supervised feature learning and extraction, which we used them for the same feature extraction layer, and for the unsupervised feature extraction layer, we used a pre-trained convolutional neural network, which is an unsupervised feature learning method as it is explained in chapter 2. In our research, we have used these both categories to see how hybrid forms of these two categories can improve the accuracy of the classification of age and gender on facial images.

3.6 Supervised Feature Extraction Layer

As explained earlier in chapter two, supervised feature extraction methods usually use a pre-trained model that is trained with well-known labeled features. These features are usually marked manually, and a machine learning model is employed to train a model to track the same features automatically later.

3.6.1 Action Units Features Extraction

In chapter two, we explained that Action Units or AUs are units of a coding system for facial expressions. As it is explained in (Hess, Adams Jr, & Kleck, 2004), the gender of each individual has a big impact on the rate of facial expressions. In other words, gender affects the intensity of angriness, happiness, and sadness. Such features like Action Units, which illustrate facial expression, are useful features for gender as well because the intensity for these features is different for each gender. Moreover, (Valstar, Jiang, Mehu, Pantic, & Scherer, 2011) and (Fabian Benitez-Quiroz, Srinivasan, & Martinez, 2016) have used Action Units as very good features for facial expression recognition.

Another research that is done by (H.-Y. Huang, 2009) demonstrated that women, in general, have stronger facial expressions and stronger experiences of emotion than men. Moreover, women generally show more intense facial expressions than men, despite

having similar emotional experiences. Compared with men, women showed notably stronger facial expression during the happiness and fear conditions, but not during the anger condition.

So we use Action Units because, as we mentioned, base on the research from (H.-Y. Huang, 2009) and (Hess et al., 2004), the intensity of each facial expression is affected by gender, so we use features which illustrating facial expression such as AUs to help classifier to classify gender.

3.6.2 HOG Features Extraction

In chapter two, we explained how the Histogram of Oriented Gradients or HOG works in detail. HOG is one of the robust handcrafted features from facial images that can be used for age or gender classification. For example, (Azzopardi, Greco, & Vento, 2016) achieved 92.6% accuracy for gender recognition using HOG features. The analysis over age estimation in paper (Huerta, Fernández, Segura, Hernando, & Prati, 2015) for different techniques shows that the fusion of HOG with other useful features can improve the accuracy of age classification significantly. So we use HOG as another group of handcrafted facial features besides other feature groups.

3.7 Implementation of AUs and HOG

We have used OpenFace, which is an open-source library that is implemented by (Baltrušaitis, Robinson, & Morency, 2016). OpenFace has a robust implementation of HOG and AUs. In a supervised feature extraction layer, we have two independent feature extraction modules. Each module extract respected features and feeds to the next layer. We have implemented an interface to extract AUs and HOG features in realtime.

We have implemented a python wrapper for the C++ code of the OpenFace library. In that wrapper which is a python, C and C++ code we have implemented a class called

HCFeatures which implements a function called **Extract_AUs_HOG**. This function has an argument *arr*, which is a matrix, or simply it is just an image. This function returns a 2D vectors of vectors which contains AUs and HOG features.

Inside the function in the wrapper code (**HCFeatures::Extract_AUs_HOG**) we use an instance initialized inside **HCFeatures::Load()** of another class which is called **OpenFaceInterface**. This instance is used to load pre-trained model and to extract features. **OpenFaceInterface** implements a function with same name **Extract_AUs_HOG** which basically does face detection and then detects facial landmarks and uses those landmarks to extract AUs and HOG features using method **StaticAUs_And_HOG** from OpenFace (Baltrušaitis et al., 2016).

3.8 Unsupervised Feature Extraction Layer

In this layer, we have used a pre-trained convolutional neural network to extract unsupervised features. To do so, we have employed a CNN that is introduced by (Levi & Hassner, 2015). The network is trained with thousands of facial images. In other words, we have trained this network independently, and once weights are adjusted enough, then this network without a classifier layer is used as a feature extractor. The CNN network by (Levi & Hassner, 2015) consists of multiple layers. In this part, we want to explain the architecture for the pre-trained CNN in detail.

3.8.1 Pre-Trained CNN

This CNN model is pre-trained with thousands of facial images by (Levi & Hassner, 2015). In other words, this CNN model has well-adjusted weights for convolutional layers that we can reuse for the unsupervised feature extraction layer.

3.8.2 Architecture of the Pre-trained CNN

The CNN model includes three convolutional layers following with two fully-connected layers, including a small number of neurons. All three channels from the colorful input image are processed directly through the network. At first, images are resized to 256×256 and cropped of 227×227, then is fed to the network. Convolutional layers mentioned above are described as below:

- **Layer 1:** 96 filters where each filter has the size of $3 \times 7 \times 7$ pixels are employed to the input in the very first layer which then followed by ReLu units and then a max-pooling layer taking the maximal value of 3×3 regions with two-pixel strides and also a local response normalization layer.

- **Layer 2:** The $96 \times 28 \times 28$ output of the prior layer is then processed by the following convolutional layer, containing 256 filters of size $96 \times 5 \times 5$ pixels. Again, this is also followed by ReLU, a max-pooling layer, and a local response normalization layer with the same hyper parameters as before.

- **Layer 3:** Finally, the third and last convolutional layer operates on the $256 \times 14 \times 14$ blob by applying a set of 384 filters of size $256 \times 3 \times 3$ pixels, followed by ReLU and a max-pooling layer.

The following fully connected layers are then defined by:

- **Layer 4:** A first fully connected layer that takes the output of the third convolutional layer and contains 512 neurons, followed by a ReLu and a dropout layer.

- **Layer 5:** A second fully connected layer that takes the 512-dimensional output of the first fully connected layer and again contains 512 neurons, followed by a ReLu and a dropout layer.

- **Layer 6:** A third, fully connected layer that outlines the final classes for age or gender.

Lastly, the output of the very last layer is fed to the classifier, which is a Softmax layer that specifies a probability for each class. The maximum probability among output probabilities will be the predicted class or the input test image.

3.8.3 CNN Features

We have checkpoints that stores final weights for the trained CNN model, as we mentioned earlier, the classifier layer will be skipped from this network, so only convolution and pooling layers are used for CNN feature extracting layer. To do so last two fully-connected layers and classifiers are dropped, and we only need the output from the final convolutional layer, so we reload stored weights from checkpoints that are created during the training process of the pre-trained CNN model into respected filters of the CNN feature extractor. In other words, we reuse weights for filters of convolutional layers from the pre-trained CNN model for our CNN feature extractor. The structure of CNN layers is illustrated in Figure 3.2. The output from the last max-pooling layer (pool3) is flatted, which contains unsupervised features that we feed to the next layer, the same as the supervised feature extraction layer.

In table 3.1 we have illustrated parameters of the CNN feature extractor network that is fine-tuned with the pre-trained CNN model.

3.9 Feature Fusion and Dimensionality Reduction Layers

In this step, we are trying to combine outputs from feature extraction units in the feature extraction layer. Our goal is to append HOG, AUs, and CNN features. The problem is we can not append these features directly together, and the main reason for that is all these feature groups are in very different dimensions. For action unit features, the feature

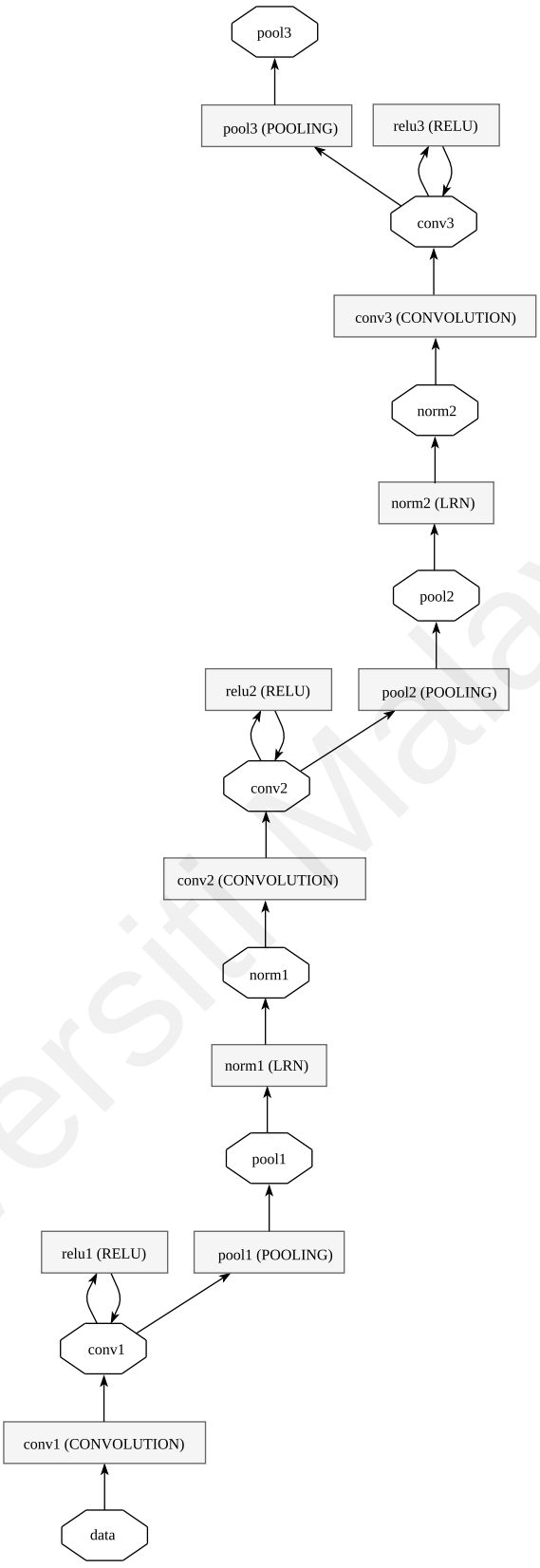

Figure 3.2: Overview of CNN feature extractor

has a dimension of 34, and the HOG features vector contains 4464 decimal values, and finally, the highest dimension goes for output from the CNN model, which is 13824 length

Table 3.1: CNN Parameters

Name	Type	Parameters
conv1	convolution 2d	Input: facial images (227×227) Number of output filters: 96 Kernel Size: [7, 7] Stride: [4, 4] Padding: VALID
pool1	max pooling 2d	Input: conv1 Kernel Size: [3, 3] Stride: [2, 2] Padding: VALID
norm1	local response normalization	Input: pool1 Depth radius: 5 Alpha: 0.0001 Beta: 0.75
conv2	convolution 2d	Input: norm1 Number of output filters: 256 Kernel Size: [5, 5] Stride: [1, 1] Padding: VALID
pool2	max pooling 2d	Input: conv2 Kernel Size: [3, 3] Stride: [2, 2] Padding: VALID
norm2	local response normalization	Input: pool2 Depth radius: 5 Alpha: 0.0001 Beta: 0.75
conv3	convolution 2d	Input: norm2 Number of output filters: 384 Kernel Size: [3, 3] Stride: [1, 1] Padding: SAME
pool3	max pooling 2d	Input: conv3 Kernel Size: [3, 3] Stride: [2, 2] Padding: VALID
output	reshaped vector	Size: $384 \times 6 \times 6 = 13824$

features vector. Combining such features without any adjustment most likely will lead to overfitting issues, or in the best case, features like AUs will be useless because it will have a minor impact in the combined feature vector. Another problem is that these features in total are so vast, and they consume a lot of unnecessary resources, and because we are dealing with extensive datasets, it can lead to a prolonged and unnecessary process.

To overcome this issue, we have mapped our raw feature vectors into three independent autoencoders, which lead to features vector of $256 \times 3 = 768$ dimension, which is an input for our classifier.

The training process for each autoencoder happens individually. When each autoencoder is trained, then all autoencoders can encode features reasonably, and then we use them without training them again. We evaluate our autoencoders with Mean Squared Error (MSE) metric. In other words, our autoencoder trainer tries to minimize MSE. We are using the RMSProp algorithm as the optimizer to reduce the MSE. The difference between RMSProp (Ruder, 2016) and gradient descent algorithm is how gradients are calculated. Each neuron in hidden layers of this network has equipped with a sigmoid activation function.

3.9.1 Autoencoder for AUs

The first autoencoder that we are explaining in this section is employed to encode 34 Action Units features vector into a higher dimension because, as it is mentioned earlier, combining small feature vectors with big feature vectors will most likely lead to overfitting problem. As it is illustrated in Figure 3.3, the autoencoder has an input layer with 34 neurons and the first hidden layer with 512 neurons and the second hidden layer with 256 neurons these two hidden layers are for encoder we have the same size hidden layers for the decoder. So the third hidden layer is again with 256 neurons and fourth hidden layer with 512 neurons and finally 34 output neurons. This autoencoder tries to reconstruct input to the output layer.

To train the autoencoder for AUs, we used an initial learning rate of 0.01 with a batch size of 128 and a total epoch of 400. All weights and biases initialized with random values from a normal distribution. Then these values are adjusted using the RMSProp algorithm until the MSE reaches global minima. Once the network training is finished, we use a

forward function to get encoded features only. To do so, we feed the AUs features vector to the network, and values from sigmoids of the second hidden layer of the encoder are reduced or encoded features.

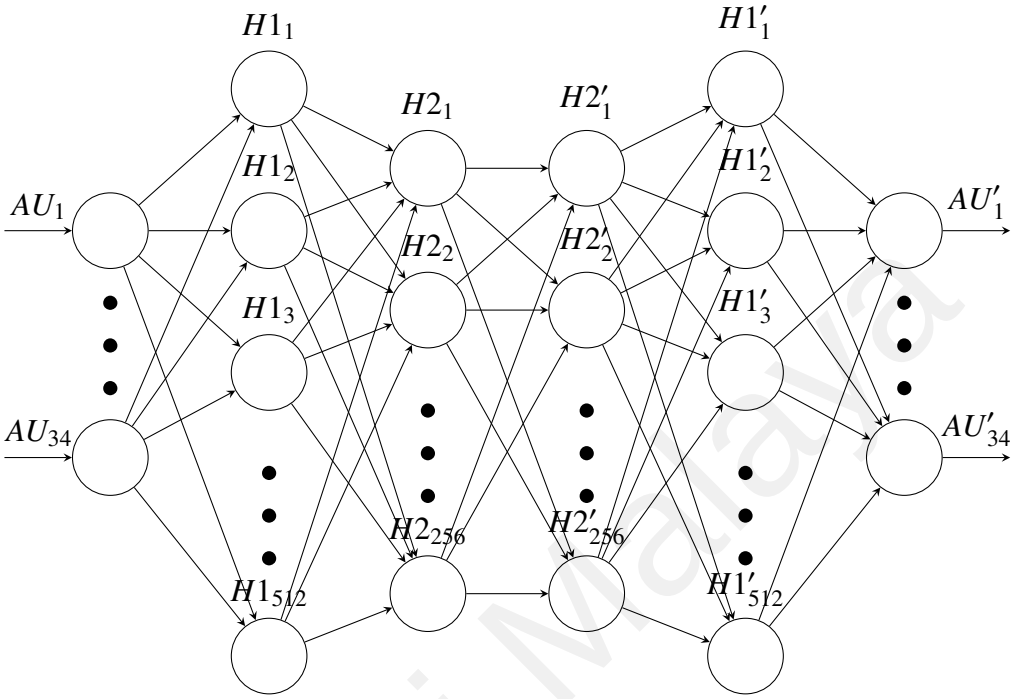

Figure 3.3: Autoencoder to encode AUs features vector

3.9.2 Autoencoder for HOG

The second autoencoder that we are explaining in this section is employed to encode 4464 HOG features vector into a lower dimension. As it is illustrated in Figure 3.4, the autoencoder has an input layer with 4464 neurons and the first hidden layer with 512 neurons and the second hidden layer with 256 neurons these two hidden layers are for encoder we have the same size hidden layers for the decoder. So the third hidden layer is again with 256 neurons and fourth hidden layer with 512 neurons and finally 4464 output neurons. This autoencoder tries to reconstruct input to the output layer.

To train the autoencoder for HOG, we used an initial learning rate of 0.01 with a batch size of 128 and a total epoch of 5000. All weights and biases initialized with random values from a normal distribution. Then these values are adjusted using the RMSProp

algorithm until the MSE reaches global minima. Once the network training is finished, we use a forward function to get encoded features only. To do so, we feed the HOG features vector to the network, and values from sigmoids of the second hidden layer of the encoder are reduced or encoded features.

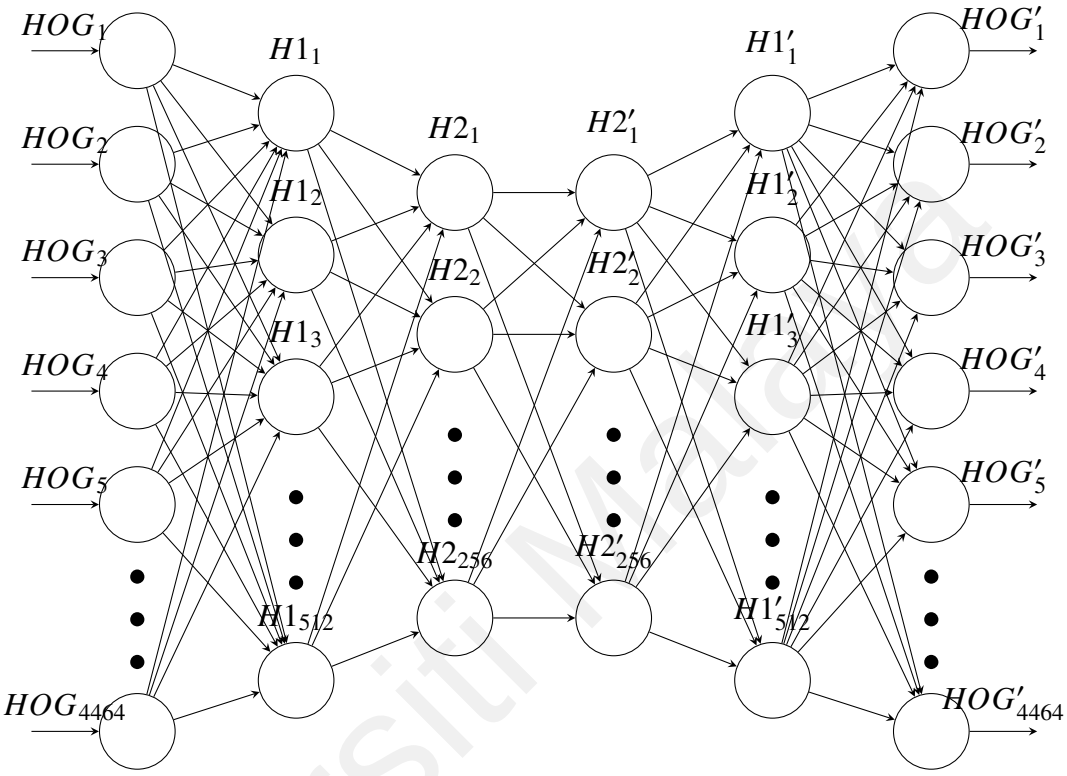

Figure 3.4: Autoencoder to encode HOG features vector

3.9.3 Autoencoder for CNN

The last autoencoder that we are explaining in this section is employed to encode 13824 CNN features vector into a lower dimension. As is illustrated in Figure 3.5, the autoencoder has an input layer with 13824 neurons and the first hidden layer with 512 neurons and the second hidden layer with 256 neurons these two hidden layers are for encoder we have the same size hidden layers for the decoder. So the third hidden layer is again with 256 neurons and fourth hidden layer with 512 neurons and finally 13824 output neurons. This autoencoder tries to reconstruct input to the output layer.

To train the autoencoder for CNN, we used an initial learning rate of 0.01 with a

batch size of 128 and a total epoch of 5000. All weights and biases initialized with random values from a normal distribution. Then these values are adjusted using the RMSProp algorithm until the MSE reaches global minima. Once the network training is finished, we use a forward function to get encoded features only. To do so, we feed the CNN features vector to the network, and values from sigmoids of the second hidden layer of the encoder are reduced or encoded features.

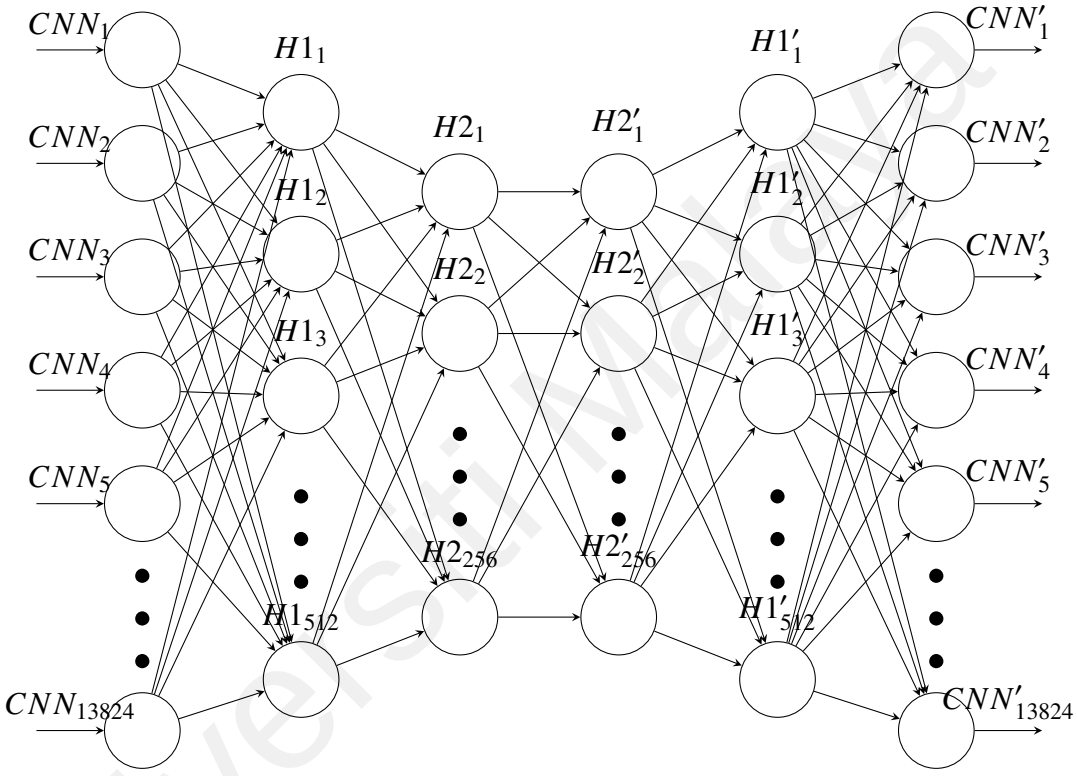

Figure 3.5: Autoencoder to encode CNN features vector

3.9.4 Combining Features

In this section, we explain how all encoded features from previous sections are combined. As explained in sections 3.9.1, 3.9.2 and 3.9.3 we can encode our features from vastly different dimensions into same dimensions. As a result, we have three encoded features vectors with a size of 256. As is illustrated in Figure 3.6, we append these three features vectors together, and as a result, we will have a single features vector of 768. Later in the next section, we will use this features vector as input feed to our classifier.

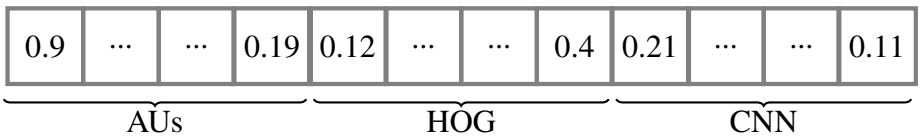

Figure 3.6: Feature fusion

3.10 Classification Layer

In this layer, we are attempting to predict and classify age and gender using the final feature map from the previous layer. There are a couple of famous and well-known classifiers to solve this problem. As it is explained earlier, Softmax is one of the widely used solutions for multi-class classification, which is a generalized form of logistic regression. Logistic regression works as a binary classifier within it and generalizes binary classifiers to multi-class classification.

As we discussed in previous chapters, the main problem with the Softmax algorithm is that it will not work if our data is not linearly separable and because deep network feature extraction methods (in our case CNN) can not guarantee linear separability of features, so Softmax is not a right choice for us. In this research, we are investigating our proposal classifier H-HP-ELM to overcome limitations of Softmax. This solution has one crucial benefit, H-HP-ELM makes it possible to use our proposed network in a parallel mode, which for deep learning modules, it can be an advantageous approach.

The mathematics behind H-ELM has been explained in the previous chapter. So far, the only implementation of H-ELM is in Matlab code, and HP-ELM is implemented in python. So we have to re-implement H-ELM in Python to be able to do our research about the performance of H-HP-ELM for age and gender classification.

If the classification layer is considered as an independent layer, the input would be encoded and appended features vector (which the dimension is 768) from the previous section for all or batched training or evaluation dataset. In other words, as it is illustrated

in Figure 3.7, we build a matrix of $M \times 768$ where M is the number of facial images in the respective dataset, and we feed it to the classifier.

$$\begin{bmatrix} H2_{AU_{1,1}} & \dots & H2_{AU_{1,256}} & H2_{HG_{1,1}} \dots & H2_{HG_{1,256}} & H2_{CN_{1,1}} & \dots & H2_{CN_{1,256}} \\ \vdots & & \vdots & \vdots & \vdots & \vdots & & \vdots \\ H2_{AU_{M,1}} & \dots & H2_{AU_{M,256}} & H2_{HG_{M,1}} \dots & H2_{HG_{M,256}} & H2_{CN_{M,1}} & \dots & H2_{CN_{M,256}} \end{bmatrix}$$

Figure 3.7: Input matrix to H-HP-ELM

The proposed H-HP-ELM has its own three main layers:

- First ELM Autoencoder layer (Tang et al., 2016)
- Second ELM Autoencoder layer (Tang et al., 2016)
- HP-ELM (Akusok et al., 2015)

3.10.1 ELM Autoencoder

One of the ELM theory use case other than SLNFs is for autoencoders (Tang et al., 2016). Mathematically, the autoencoder maps the input data x to a higher level representation, and then uses latent representation y through a deterministic mapping $y = h_\theta(x) = g(A \cdot x + b)$, parameterized by $\theta = A, b$, where $g(\cdot)$ is the activation function, A is a $d\prime \times d$ weight matrix and b is a bias vector. The resulting latent representation y is then mapped back to a reconstructed vector z in the input space $z = h_{\theta'}(y) = g(A' \cdot y + b)$ with $\theta' = \{A', b'\}$ (Tang et al., 2016). The pseudo code in Algorithms 1 and 2 is the implementation of the ELM-Autoencoder.

The function $sparse\_elm\_autoencoder$ is employed to extract sparse features of the input data from the previous layer or input in each layer as it is explained in (Tang et al., 2016) in detail,. 2-layer unsupervised learning is performed to obtain sparse high-level features eventually. As it is explained in (Tang et al., 2016) $N1$ and $N2$ are values

that are required to build $b1$ and $b2$ matrices, which are used for first and second ELM-Autoencoders sequentially. Base on our experimental results, we found that $N1 = 768$ and $N2 = 768$ are suitable values for our use case. Parameters $b1$ and $b2$ are matrices that have random initial values, which $b1$ is the bias for first ELM-Autoencoder, and $b2$ is the bias for the second ELM-Autoencoder. $b1$ has a shape of $(I + 1) \times N1$, where I is the input dimensions for the classifier, which equals the features vector size from the previous section. $b2$ has a shape of $(N1 + 1) \times N2$ (Tang et al., 2016). Both ELM-Autoencoders have an iterative loop of 100 times to calculate $\beta 1$ for the first sparse autoencoder and $\beta 2$ for second sparse autoencoder. $\beta 1$ and $\beta 2$ are required for the feed-forward network that is used for evaluation and testing later, which are output weights for each autoencoder. The λ for both autoencoders are 10^{-3}.

Algorithm 1 Sparse ELM Autoencoder

```
function SPARSE_ELM_AUTOENCODER(A, b, λ, iterations)
    AA = (A') * A;
    Lf = max(eigen(AA));
    Li = 1/Lf;
    alp = λ * Li;
    m = size(A,2);
    n = size(b,2);
    x = zeros(m,n);
    yk = x;
    tk = 1;
    L1 = 2 * Li * AA;
    L2 = 2 * Li * A' * b;
    for i=1:iterations do
        ck = yk − L1 ∗ yk + L2;
        x1 = (max(abs(ck) − alp, 0)). ∗ sign(ck);
        tk1 = 0.5 + 0.5 ∗ sqrt(1 + 4 ∗ tk^2);
        tt = (tk − 1)/tk1;
        yk = x1 + tt ∗ (x − x1);
        tk = tk1;
        x = x1;
    return x, tk
```

Function $feedforward\_L1\_L2$ in Algorithm 3 is for feedforward usage, which means whenever the model is trained, this can be used to extract unsupervised features from the input of the classification layer.

Algorithm 2 First and Second ELM Autoencoders

1: **function** LAYERS_L1_L2($input, b1, b2, iter1, iter2$)
2: $input = zscore(input^T)^T$;
3: $H1 = [input.1 * ones(size(input, 1), 1)]$;
4: $A1 = H1 * b1$;
5: $A1 = mapminmax(A1)$;
6: $\beta 1 = sparse\_elm\_autoencoder(A1, H1, 10^{-3}, 50)^T$;
7: $T1 = H1 * \beta 1$;
8: $T1, ps1 = mapminmax(T1^T, 0, 1)$;
9: $T1 = T1^T$;
10: $H2 = [T1.1 * ones(size(T1, 1), 1)]$;
11: $A2 = H2 * b2$;
12: $A2 = mapminmax(A2)$;
13: $\beta 2 = sparse\_elm\_autoencoder(A2, H2, 10^{-3}, 50)^T$;
14: $T2 = H2 * \beta 2$;
15: $T2, ps2 = mapminmax(T2^T, 0, 1)$;
16: $T2 = T2^T$;
17: **return** $\beta 1, \beta 2, T2, ps1, ps2$

Algorithm 3 Feed-Forward Layer 1 and Layer 2

1: **function** FEEDFORWARD_L1_L2(test_x, $\beta 1$, $\beta 2$, ps1, ps2)
2: $test\_x = zscore(test\_x^T)^T$;
3: $HH1 = [test\_x\ 0.1 * ones(size(test\_x, 1), 1)]$;
4: $TT1 = HH1 * \beta 1$;
5: $TT1 = mapminmax(TT1^T, ps1)^T$;
6: $HH2 = [TT1\ 0.1 * ones(size(TT1, 1), 1)]$;
7: $TT2 = HH2 * \beta 2$;
8: $TT2 = mapminmax(TT2^T, ps2)^T$;
9: **return** $TT2$

The whole process of extracting features using ELM-Autoencoder and finally classifying them with HP-ELM has been shown in Algorithm 4. With a closer look at the pseudo-code in Algorithm 4 it can be seen that from lines 1 to 12, we train the classifier and from lines 13 to 17, we evaluate the model. Variables $\beta 1, \beta 2, ps1$ and $ps2$ needs to be stored for later to be used as parameters for classification pipeline (to predict). The *elm* variable is also stored as a serialized object in the filesystem, which is an instance of ELM from *hpelm* library, which is implemented by (Akusok et al., 2015) as we explained earlier. Having the stored parameters and elm object, we can use this trained class for classification later. The HP-ELM network that we are using needs 3000 initial neurons, which later will use the Leave-one-out validation method to select the optimal

Algorithm 4 Hierarchical High Performance Extreme Learning Machine

```
1: N1 = 768;
2: N2 = 768;
3: b1 = rand(input.shape[1] + 1, N1) − 1.0;
4: b2 = rand(N1 + 1, N2) − 1.0;
5: β1, β2, T2, ps1, ps2 = layers_L1_L2(input, b1, b2, iter1, iter2);
6: elm = ELM(T2.shape[1], target.shape[1]);
7: elm.add_neurons(3000, 'rbf_l2');
8: if (regression) {
9: ... elm.train(input, target, 'LOO', 'r');
10: } else {
11: ... elm.train(input, target, 'LOO');
12: }
13: predict = elm.predict(input);
14: if (regression) {
15: ... regression_predict = results_regression(predict);
16: ... regression_target = results_regression(target);
17: ... training_accuracy = accuracy(regression_predict, regression_target);
18: } else {
19: ... argmax_predict = results_argmax(predict);
20: ... argmax_target = results_argmax(target);
21: ... training_accuracy = accuracy(argmax_predict, argmax_target);
22: }
23: TT2 = feedforward_L1_L2(test_input, β1, β2, ps1, ps2);
24: test_predict = elm.predict(TT2);
25: if (regression) {
26: ... test_regression_predict = results_regression(test_predict);
27: ... test_regression_target = results_regression(test_target);
28: ... test_accuracy = accuracy(test_regression_predict, test_regression_target);
29: } else {
30: ... test_argmax_predict = results_argmax(test_predict);
31: ... test_argmax_target = results_argmax(test_target);
32: ... test_accuracy = accuracy(test_argmax_predict, test_argmax_target);
33: }
```

regularization parameter. The leave-one-out cross-validation (LOO) approach is one of the most effective methods for model selection and parameter optimization in machine learning (van Heeswijk & Miche, 2015). We use regression for age estimation, which is recognized with parameter *r* in algorithm 4 in line 9 because for age estimation we do not have any class, but we predict the actual age value instead of class, and we need that for benchmark evaluation that we will explain in next chapter.

Speaking of datasets and desired labels that we need in our research, we have two classes for gender classification and three classes for age classification. All of the datasets

of facial images that are used for classification in this research have labels as values in table 3.2 and for age estimation where we estimate age value (and not the category) we have age value (numerical e.g., 25 or 57) as the label or target value.

Table 3.2: Classes used in our classification

Type	Class	Label
Gender classification	Female	0
	Male	1
Age classification	Non-adult	0
	Adult	1
	Elderly people	2

3.11 Softmax vs H-HP-ELM

As we discussed Softmax and its' limitations and to achieve enough results to justify our objectives, we want to evaluate the performance of our classification layer. Since we want to compare our classifier with Softmax, we have used Softmax as the classifier for an independent model, which is identical to our architecture in all other layers, and the only difference is the classification layer, which we replaced H-HP-ELM with Softmax. We have used the same datasets that we have used for our proposed architecture to evaluate our models.

Usually, a Softmax classifier comes with one or multiple fully connected layers. Both H-HP-ELM and Softmax (plus fully connected layers) have the same inputs and outputs, but they are trained in different ways, as we explained earlier in previous and current chapters. In this research, we decided to use two fully connected layers, followed by a Softmax classifier.

The classification layer that we have chosen contains the input layer, which is the output from autoencoders and one fully connected layer, and finally, a Softmax classifier (Figure 3.8). $C_1, C_2, ...C_n$ represents the classes of the Softmax classifier.

We need a loss function and an optimizer to train our classifier. We use Softmax

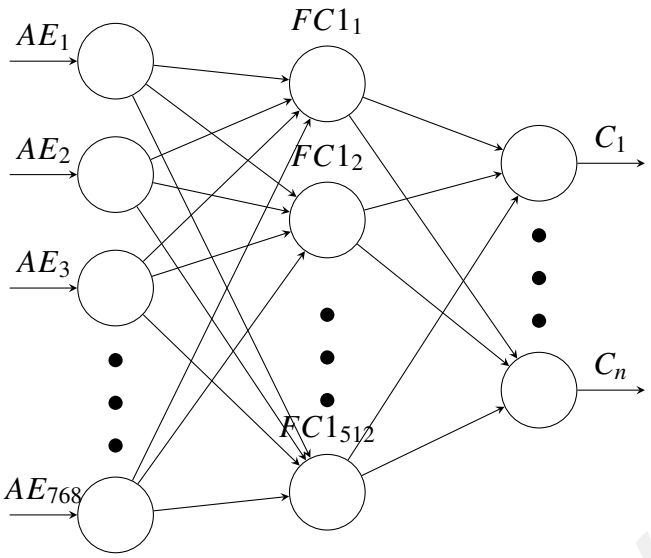

Figure 3.8: Fully connected layer + Softmax classifier

cross-entropy loss function in combination with the exponential moving average technique with an initial decay of 0.9. We have used an exponential staircase decay momentum optimizer with an initial value of 0.9 with a decay rate of 0.5, which in every 5k steps cuts the learning rate in half.

3.12 Summary

To summarize, in our classification layer, we are feeding the output features that we extracted in feature extraction layer to the H-HP-ELM classifier. So our feature, which has 768 dimensions, will be changed to more sparse features after passing through hierarchical layers of H-HELM, which means only significant and essential features are selected. The final feature size in the final layer before the HP-ELM classifier is 400, which has been chosen based on an experiment. These features are the very final features that are suitable features to be used for classification. It is then fed into the HP-ELM classifier, which our HP-ELM classifier contains initially 3000 neurons in the hidden layer, which are radial basis function neurons following having ℓ_2 normalization. Then we are selecting the optimal number of neurons using Leave-one-out or LOO while training our network.

CHAPTER 4: EXPERIMENTS

4.1 Introduction

In this chapter, we explain how our experiments are established. We will discuss details that are not covered in the previous chapter. As an example, we will explain how our research is built, what environment we have used, and how the datasets are prepared. We will also discuss the method that we use to evaluate the efficiency of our proposed method.

4.2 Autoencoder Dataset

To train our autoencoders, we needed an independent dataset of facial images that are not used for the classification and estimation layer. To do so, we have used a portion of the WFLW dataset (Wu et al., 2018) with 6551 facial images from the wild. We have moved out pictures with a bad pose or low-quality images, which brought us 5017 facial images. Then we have randomly chosen 3000 facial images from these remained images to train our autoencoders. We have extracted CNN, AUs, and HOG features for all these images, and we stored them as binary files with the same naming as the original image file name but with different extensions. Later we used these binary files, which contain features to train our autoencoders that we explained in the previous chapter. This dataset is not used for classifier training step.

4.3 Classification and Estimation Datasets

We used multiple facial images benchmark datasets to evaluate our proposed method. These datasets are used in various researches related to facial image analysis. They are publicly available, and for a few of them, it is required to sign up, which the process is straightforward.

4.3.1 FG-NET Dataset

FG-NET dataset is introduced by (Lanitis & Cootes, 2002), which contains 1002 images from 82 individuals. Ages ranging from newborns to 69 years old. The Leave One Person Out (LOPO) protocol is the commonly used evaluation protocol for FG-NET. All samples of a single person are employed as the test set, and the rest of the other 81 persons are employed for the train set. This process is repeated for all 82 persons on the FG-NET dataset. This evaluation protocol makes sure that images of a single person are not employed in the testing and training set at the same time to avoid any bias for the trained models. In our experiment, we use the same protocol to evaluate our model with the benchmark dataset. The FG-NET dataset is so small to train our model, so we only use this dataset to test age estimation and not for training.

4.3.2 LAP Dataset

The ChaLearn Looking at People or LAP dataset, which is introduced in (Escalera et al., 2015) contains 4,699 images labeled with real and apparent age. The dataset is split into a 2,476 train set, 1,136 validation set, and 1,087 test set. The LAP dataset is not big enough to train our model, so we only use this dataset to test age estimation and gender classification but not for training step. Distribution of ages in the LAP dataset is from 20 to 40 years but has a small number of images per year for ages from 0 to 15 and from 65 to 100. This dataset is used for benchmark age estimation evaluation of our model.

4.3.3 CACD Dataset

The Cross-Age Celebrity Dataset or CACD (B.-C. Chen, Chen, & Hsu, 2014) contains 163,446 images of 2,000 celebrities from 2004 to 2013. The age is estimated using the query year and the known date of birth. The dataset splits into training, validation, and test sets. Images from 1800 celebrities are used as a train set, 80 as the validation set,

and 120 as the test set. As it is self-explanatory, this dataset is used for benchmark age estimation evaluation of our model.

4.3.4 LFW Dataset

The Labeled Faces in the Wild or LFW dataset (G. B. Huang, Ramesh, Berg, & Learned-Miller, 2007) contains 13,233 photos of 5,749 individuals and 1,680 people with two or more images that were collected in 2007. LFW is one of the standard benchmarks for gender classification. The protocol for the LFW dataset is to use 10-fold cross-validation to evaluate the performance on the dataset. However, the training set (nine folds) is imbalanced in terms of the number of samples for each gender. To fix that issue, we combined training set with WIKI-IMDB datasets, and we also performed downsampling for Male class and oversampling for Female class.

Table 4.1: Statistics of original LFW dataset

Male	10,268
Female	2,966
Total	13,233

4.3.5 MORPH-II Dataset

MORPH-II dataset is introduced by (Ricanek & Tesafaye, 2006). This dataset contains 55,134 images of 13,618 individuals age range from 16 to 77. Benchmark evaluation for this dataset is done by measuring Mean Average Error (MAE) for age estimation and Accuracy for gender classification. The labels for each sample includes age, Ethnicity, and gender. MORPH-II dataset has an uneven distribution of samples for Male (80%), and Female (20%), the performances measured on these benchmarks are prone to be biased. The same problem exists for the distribution of ages. The first benchmark evaluation for MORPH-II is to divide the dataset into an 80% training set and 20% validation or test set. A more efficient protocol for this dataset introduced by (Guo & Mu, 2010),

which later adopted by many other researchers. The MORPH-II dataset is split into three non-overlapping parts S_1, S_2 and S_3. gender classification and age estimation models are trained using S_1 and validated by S_2 and S_3, and again trained on S_2 and validated on S_1 and S_3. Final results are the mean of Accuracy and MAE over these two experiments.

Regarding the age classification model, we need to evaluate the performance of our model for three categories of Non-Adults, Adults, and Elderly People. So we regrouped the MORPH-II dataset. Here are the statistics for the MORPH-II dataset that we need for model evaluation used in Table 4.2.

Table 4.2: Statistics of original MORPH-II dataset for gender and 3 categories classification

	Non-Adults	Adults	Elderly People	Total
Male	6,638	36,525	3,482	46,645
Female	831	7,207	451	8,489
Total	7,469	43,732	3,933	55,134

4.3.6 WIKI-IMDB Dataset

WIKI dataset first used by (R. Rothe et al., 2016) and (R. Rothe, Timofte, & Gool, 2015) for age and gender classification. This dataset contains more than 62K images. Because the dataset contains lots of low quality and bad angle photos of individuals too, so we have extracted only useful facial photos for our experiment. Here are the statistics for the WIKI dataset that we have used in Table 4.3.

Table 4.3: Statistics of WIKI dataset for gender and 3 categories classification

	Non-Adults	Adults	Elderly People	Total
Male	106	3,493	525	4,124
Female	124	2,290	228	2,642
Total	230	5,783	753	6,766

(R. Rothe et al., 2016) also uses the IMDB dataset and (R. Rothe et al., 2015) for age and gender classification. This dataset contains more than 460K images. Since the dataset contains lots of low quality and bad angle photos of individuals too, so we have cleaned

up the dataset to have useful facial photos for our experiment. Here are the statistics for the IMDB dataset that we have used in Table 4.4.

Table 4.4: Statistics of IMDB dataset for gender and 3 categories classification

	Non-Adults	Adults	Elderly People	Total
Male	301	11,234	1,350	12,885
Female	582	13,944	650	15,176
Total	883	25,178	2,000	28,061

For both datasets, we extracted CNN, AUs, and HOG features for all images for both validation and training sets, and we stored them as binary files with the same naming as the original image file name but with different extensions. The stored features are used for another experiment to train our H-HP-ELM classifier. As it is explained earlier, WIKI-IMDB has many low-quality images with noises that our feature extraction methods failed to extract features properly for many of the images, so we could not do a benchmark evaluation for the dataset. For that reason, we decided to use it for training purposes only.

4.3.7 Adience Dataset

Adience dataset is introduced by (Eidinger et al., 2014) for age and gender classification. Adience dataset consists of 26,580 unconstrained images of 2,284 subjects in 8 age categories that are 0-2, 4-6, 8-13, 15-20, 25-32, 38-43, 48-53 and 60+. We have used these categories for benchmark model evaluation.

The protocol for the Adience dataset to evaluate the performance of gender and age classification is to perform training and testing using 5-fold cross-validation, which is defined by (Eidinger et al., 2014). Gender labeling is a binary classification task, and for the multi-class age classification, we report mean classification error across all age groups. Moreover, for age classification, we need to test with the 1-off method that the accuracy measured considering predictions when the prediction is off only one age group as correct prediction (either younger or older group).

As it is also mentioned in (Levi & Hassner, 2015), we realized that many samples in this dataset have motion blur, occlusions, too much rotation. Since they have an essential impact on the quality of age and gender classification and estimation results, we removed them out from our dataset, and to compensate for the data loss. We used the same oversampling method as (Levi & Hassner, 2015) has used. We also flipped many samples horizontally to compensate for the data loss. We have performed oversampling only for the training set and not for the validation set to make sure the evaluation for benchmark evaluation remains valid.

Regarding the age classification model, we need to evaluate the performance of our model for three categories of Non-Adults, Adults, and Elderly People. So we regrouped samples in the Adience dataset, and here are the statistics for the Adience dataset that we need for model evaluation used in Table 4.5.

Table 4.5: Statistics of Adience dataset for gender and 3 categories classification

	Non-Adults	Adults	Elderly People	Unknown	Total
Male	4,643	5,547	837	21	11,048
Female	5,371	6,212	823	1,277	13,683
Total	10,014	11,759	1,660	1,298	24,731

4.3.8 M3C Dataset

As we explained earlier in this chapter, datasets used for our tests are not balanced in terms of classes. For example, as it is illustrated in table 4.4, the number of old females in the IMDB dataset is fewer than adult females as another example number of old males is considerably less than adult males in the MORPH-II dataset (table 4.2). This condition can harm the training process for samples with minority class or samples with the majority class.

We wanted to do an experiment with a more balanced data by oversample minority class and undersample majority class. We also tried to combine these datasets and

measure the impact of the model that is trained using this balanced and merged dataset. For undersampling, we randomly removed facial images from majority classes, and for oversampling we randomly selected facial images from minority classes and made copy from them and to avoid overfitting due to duplicated data and possible bias to the model, we rotated images for 1 to 3 degrees clockwise and anticlockwise or/and we flipped them as well. Although the data is still imbalanced after oversampling and downsampling, the imbalance rate has improved significantly. For example as it is illustrated in tables 4.2 and 4.6, for the merged dataset we have 53.7% Male and 46.3% Female while for MORPH-II dataset we have 15.4% Female and 84.6% Male. We have merged our datasets in order to balance our training sets as much as possible and to do so, we have combined all samples together considering the class names as the directory names to separate based on classes, in other words, the directory structure is the same as before but all files from all datasets that we have for same class are gathered in a single directory with same class name. These files are the same files that we generated for each dataset (*.hog, *.aus and *.cnn), which are handcrafted and unsupervised features that we explained earlier in the previous chapter. These features are used for our experiment on the merged dataset to train our H-HP-ELM classifier. Later we evaluated our model using 10-fold cross-validations to evaluate our model. We call this dataset the M3C dataset in this research for convenience.

Table 4.6: Statistics of the M3C dataset for gender and 3 categories classification

	Non-Adults	Adults	Elderly People	Total
Male	11,688	14,200	12,388	38,276
Female	13,816	14,827	4,304	32,947
Total	25,504	29,027	16,692	71,223

4.3.9 M-Ages Datasets

As we explained earlier, some of the benchmark datasets in previous sections are too small to be used as a training set. However, we still need them for benchmark evaluation. So

we merged LAP, CACD, WIKI-IMDB, and MORPH-II datasets into a single training set. For example, as it is illustrated in table 4.7, to evaluate the LAP dataset, the test set of LAP dataset is used as the test set and the training set will be the combination of CACD, WIKI-IMDB, and MORPH-II in addition to the LAP training set. However, all of the noisy, blurry images caused by motion blur, too much rotation are removed out of the merged training set. We also tried to perform undersampling and oversampling to balance and also to compensate for data loss in these M-Ages training datasets.

We call this dataset M-Ages datasets in this research, which are used to evaluate age estimation on FG-NET, LAP, CACD, WIKI-IMDB, and MORPH-II using MAE metric. For each test, we follow the respective protocol as it is illustrated in table 4.7, which are explained earlier in this chapter.

Table 4.7: M-Ages Datasets merge

	Test set	Entire dataset	Training set only	Protocol
#1	FG-NET	LAP, CACD, WIKI-IMDB, MORPH-II	-	LOPO
#2	LAP	CACD, WIKI-IMDB, MORPH-II	LAP	It has test set
#3	CACD	LAP, WIKI-IMDB, MORPH-II	CACD	It has test set
#4	MORPH-II	LAP, CACD, WIKI-IMDB	MORPH-II	S1, S2 and S3

4.4 Environment

For our experiments, we used a high-end GPU Machine, which is equipped with a Single GPU with 24GB GDDR5 Memory, Model Quadro M6000, accompanied by it with 128 GB Ram and running under Intel Xeon CPU E5-2699 with 22 Core, running 44 Threads. This Machine has Linux distribution Ubuntu version 17.10 LTS as the operating system with Python 3. We used this Machine to re-train our pre-trained CNN. Our proposed model is trained on another Machine, which is a Lenovo Y50-70 Laptop with a Core-i7 processor and 16GB of RAM.

4.5 Data Split for 3 Age Categories and Gender Classification

A common way is to, for example, split a dataset into a 70% training set and 30% test set and calculate accuracy on the test set. Researches show that dividing the dataset into training and test in this way is not an efficient approach. One of the efficient ways is to use cross-validation. To do so, we use k-fold cross-validation to split our datasets into validation and training sets. By employing k-fold cross-validation, we shuffle then split the data into k equal portions. One portion will be used as a validation set, and the rest will be used as a training set. We run this process for k times, which in our case, k=10. However, in the case of benchmark evaluation, we use the respective protocol for each benchmark dataset and for the 3 age categories classification, and also for other datasets, we use 10-fold cross-validation.

4.6 Model Evaluation

Evaluation of our model depends on the cross-validation from the previous section that we are using in our experiment. We are using k-fold cross-validation, which split our datasets into k different folds with the same amount of samples. One of the folds is used for the test, and the rest is for training. This process is repeated for k times. In our experiment, k=10. We extracted CNN, AUs, and HOG features for all images for both validation and training sets, and we stored them as binary files with the same naming as the original image file name but with different extensions. By employing this approach, we can start over the training step without extracting features again. Later the stored features are used for the training step to find the best parameters for our H-HP-ELM classifier. We want to evaluate our model by these metrics (tables 4.10 and 4.11):

- **Precision** which illustrates the percentage of correct predictions for a class out of the number of total predictions for that class.

- **Recall** which illustrates the percentage of correct prediction for a class out of the number of actual correct results that should have been returned for that class.
- **Specificity** which illustrates the percentage of samples that are correctly not predicted as expected for a class out of the number of total classes excluding that specific class.
- **F1 Score** is a harmonic mean of recall and precision.
- **Accuracy** is the percentage of correctly predicted values for all classes out of all samples that are used for prediction.

To measure these metrics of our model, we need to make a confusion matrix for our results. Since we have ten trained models due to the k-fold cross-validation method, so we have ten confusion matrices. We sum all these ten matrices into one matrix, and this final confusion matrix will measure the evaluation metrics. A similar approach is used by (Zhou et al., 2016) to evaluate their model, which they perform 10-fold, 5-fold, and 2-fold cross-validation. They run their algorithm for 20 iterations and calculate the F-scores from the average value of confusion matrices from the 20 iterations.

In our case, since we are using 10-fold cross-validation, we will generate ten confusion matrices, and we measure the average value in the range of 0.0 to 1.0.

Table 4.8: Confusion matrix for gender classification

		Predicted	
		Male	Female
Actual	Male	True Male (TM)	False Female (FF)
	Female	False Male (FM)	True Female (TF)

The terms that are used in confusion matrix for gender classification in table 4.8 are explained as below:

Table 4.9: Confusion matrix for age classification (Multi Class)

		Predicted		
		$A1$	...	A_n
Actual	$A1$	True A_1	...	False A_n
	...	...	...	...
	A_n	False A_1	...	True A_n

- True Male (TM): actual samples of class Male correctly predicted as class Male
- True Female (TF): actual samples of class Female correctly predicted as class Female
- False Male (FM): actual samples of class Female incorrectly predicted as class Male
- False Female (FF): actual samples of class Male incorrectly predicted as class Female

The terms that are used in confusion matrix for age classification in table 4.9 are explained as below:

- True A_x (TA_x): actual samples of class A_x correctly predicted as class A_x
- False A_x (FA_x): actual samples of class A_x incorrectly predicted as class from $\{A_1, A_2, ..., A_n\}$ but not A_x.

Table 4.10: Evaluation metrics and their equations for gender classification

Equation	Details
Overall Accuracy: $A = \frac{N_C}{N_T} \times 100$	where N_C is the number of samples that are predicted correctly and N_T is the total number of samples. Although accuracy is an important information but it does not provide sufficient information to evaluate the model.
Precision: $P_M = \frac{TM}{TM+FM}$ $P_F = \frac{TF}{TF+FF}$	 P_M is precision for males P_F is precision for females
Recall: $R_M = \frac{TM}{TM+FF}$ $R_F = \frac{TF}{TF+FM}$	 R_M is recall for males R_F is recall for females
Specificity: $S_M = \frac{TF}{TF+FM}$ $S_F = \frac{TM}{TM+FF}$	 S_M is specificity for males S_F is specificity for females
F1 Score: $F_M = 2 \times \frac{P_M \times R_M}{P_M + R_M}$ $F_F = 2 \times \frac{P_F \times R_F}{P_F + R_F}$	 F_M is F1-Score for males F_F is F1-Score for females

Table 4.11: Evaluation metrics and their equations for age classification

Equation	Details
Overall Accuracy: $A = \frac{N_C}{N_T} \times 100$	where N_C is the number of samples that are predicted correctly and N_T is the total number of samples. Although accuracy is an important information but it does not provide sufficient information to evaluate the model.
Precision: $P_{A_x} = \frac{TA_x}{TA_x + \Sigma FA_x}$	P_{A_x} is precision for class A_x
Recall: $R_{A_x} = \frac{TA_x}{TA_x + \Sigma FA_{y_i}}$	R_{A_x} is recall for class A_x where y in $\{1, 2, 3, ..., n\}$ and $y! = x$
Specificity: $S_{A_x} = \frac{\Sigma TA_{y_i} + \Sigma FA_{y_i}}{\Sigma TA_{y_i} + \Sigma FA_{y_i} + \Sigma FA_{x_j}}$	S_{A_x} is specificity for class A_x where y in $\{1, 2, 3, ..., n\}$ and $y! = x$
F1 Score: $F_{A_x} = 2 \times \frac{P_{A_x} \times R_{A_x}}{P_{A_x} + R_{A_x}}$	F_{A_x} is F1-Score for class A_x

4.7 Benchmark Evaluation For Age Estimation

In this section, we explain how we perform benchmark evaluation on well-known benchmark datasets that we have used in our experiments. As we explained earlier, there are datasets that we have in our experiments which they should be evaluated using Mean Absolute Error or MAE (table 4.12).

Another metric to evaluate age estimation is the ϵ-error, which was proposed in the LAP challenge. ϵ-error is useful where no ground truth age is available, but the ages are guessed by random people. This method makes sure that wrong predictions are penalized less than others in case there are too many wrong guesses by a human for the respective sample (table 4.12).

The reason to have a different approach for these datasets is that they are labeled for age estimation, while our original model is designed to classify age into three categories. So we use this approach for age estimation only, and it is not required for age and gender classification.

Table 4.12: Evaluation metrics and their equations for age estimation

Equation	**Details**
Mean absolute error: $MAE = \frac{1}{n}\sum_{i=1}^{n}\lvert p_i - g_i\rvert$	Where: - n = Total of samples - p_i = Predicted Age - g_i = Ground-truth Age
ϵ- error: $\epsilon = 1 - e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	Final ϵ-error is the average over all samples. Ranges: - from 0 = Perfect predictions - to 1 = Completely wrong

4.8 Summary

In this chapter, we discussed the benchmark datasets that we have used in our experiments. We explained our environment setup, and we explained how we extract features from our datasets. We explained how we improved the impact of an imbalanced dataset in our experiments. We evaluated our method with the confusion matrix, cross-validation, precision, recall, specificity, f1 score, accuracy and MAE. In the next chapter, we will analyze our results and the performance of our method.

CHAPTER 5: EXPERIMENTAL RESULTS AND DISCUSSION

Our proposed architecture that we explained in detail in chapter 3 has been tested on four benchmark datasets that we explained in chapter 4. We have performed experiments on these datasets to evaluate our architecture for age and gender classification and also for age estimation. In this chapter, we will discuss our experiment on each dataset and our gained results.

5.1 Age and Gender Classification

We performed multiple experiments to evaluate our architecture for gender and age (3 categories) classification on multiple datasets which are namely LFW, MORPH-II, Adience, and M3C. The protocol that we are using to evaluate our dataset is k-fold cross-validation algorithm where $k = 10$. We shuffle our dataset, and then we divide it into ten equal portions, and one of the portions is used randomly as the validation set and the rest for the training set.

5.1.1 MORPH-II Dataset

The confusion matrices for validation set evaluation results are illustrated in tables 5.1 and 5.2 (Actual vs Predicted values) which illustrates average of 10 independent other runs with same steps in range of 0 to 1.

Table 5.1: Confusion matrix for gender classification (MORPH-II)

Actual \ Predicted	Male	Female	Recall
Male	**0.9530**	0.1410	0.9074
Female	0.0470	**0.8590**	0.9269
Precision	0.9530	0.8595	-
Specificity	0.9269	0.9074	-
F-Score	0.9296	0.8919	-

Table 5.2: Confusion matrix for 3 age categories classification (MORPH-II)

Actual \ Predicted	Non-Adult	Adult	Elderly People	Recall
Non-Adult	**0.9000**	0.0885	0.0260	0.7837
Adult	0.0820	**0.8845**	0.1910	0.9196
Elderly People	0.0180	0.0270	**0.7830**	0.8473
Precision	0.8998	0.8845	0.7828	-
Specificity	0.9677	0.8247	0.9628	-
F-Score	0.8410	0.8696	0.9014	-

5.1.2 Adience Dataset

The confusion matrix for validation set evaluation results are in tables 5.3 and 5.4 (Actual vs Predicted values) which illustrates average of 10 independent other runs with same steps in range of 0 to 1. Please note that for adience dataset those images that have unknown labels for age are removed from dataset.

Table 5.3: Confusion matrix for gender classification (Adience)

Actual \ Predicted	Male	Female	Recall
Male	**0.8969**	0.1122	0.8735
Female	0.1031	**0.8878**	0.9088
Precision	0.8969	0.8878	-
Specificity	0.9088	0.8735	-
F-Score	0.8850	0.8982	-

Table 5.4: Confusion matrix for 3 age categories classification (Adience)

Actual \ Predicted	Non-Adult	Adult	Elderly People	Recall
Non-Adult	**0.9349**	0.01515	0.0637	0.9564
Adult	0.0393	**0.95989**	0.0820	0.9456
Elderly People	0.0258	0.02496	**0.8546**	0.8468
Precision	0.9349	0.9599	0.8545	-
Specificity	0.9606	0.9626	0.9761	-
F-Score	0.9455	0.9527	0.8506	-

5.1.3 LFW Dataset

The confusion matrix for validation set evaluation results are in table 5.5 (Actual vs Predicted values) which illustrates average of 10 independent other runs with same steps in range of 0 to 1.

Table 5.5: Confusion matrix for gender classification (LFW)

Actual \ Predicted	Male	Female	Recall
Male	**0.9827**	0.0211	0.9942
Female	0.0173	**0.9789**	0.9394
Precision	0.9827	0.9789	-
Specificity	0.9394	0.9942	-
F-Score	0.9884	0.9587	-

5.1.4 M3C Dataset

In the previous chapter, we have explained how important it is to have a balanced dataset and how an imbalanced dataset can affect the process of our model training. So we decided to combine datasets, and then we perform oversampling and undersampling methods to make sure we have a roughly balanced merged dataset to train our model, as we explained in the previous chapter. Here in this section, we are explaining the results for our experiment on the merged dataset.

The confusion matrix for validation set evaluation results are in tables 5.6 and 5.7 (Actual vs Predicted values) which illustrates average of 10 independent other runs with same steps in range of 0 to 1.

Table 5.6: Confusion matrix for gender classification (Merged Dataset)

Actual \ Predicted	Male	Female	Recall
Male	**0.9401**	0.0594	0.9683
Female	0.0599	**0.9406**	0.8907
Precision	0.9401	0.9406	-
Specificity	0.8907	0.9683	-
F-Score	0.9540	0.9150	-

Table 5.7: Confusion matrix for 3 age categories classification (Merged Dataset)

Actual \ Predicted	Non-Adult	Adult	Elderly People	Recall
Non-Adult	**0.9296**	0.0323	0.0168	0.8551
Adult	0.0328	**0.9437**	0.0695	0.9856
Elderly People	0.0376	0.0240	**0.9137**	0.7327
Precision	0.9296	0.9437	0.9137	-
Specificity	0.9859	0.8410	0.9930	-
F-Score	0.8908	0.9642	0.8133	-

5.2 Benchmark Evaluation Results

In this section, we want to perform a benchmark evaluation of our proposed model. We want to evaluate how our model performs for age and gender classification and age estimation.

5.2.1 Gender Classification

We have three different datasets to evaluate our model for gender classification, which are, namely, LFW, MORPH-II, and Adience datasets. We use the M3C plus training set of each mentioned dataset as the training set and test or validation set of the selected dataset as the test or validation set following the respective protocol of each dataset that is explained in the previous chapter. In table 5.10, we have compared our results with other works that have done experiments on these datasets using the same protocol. The LFW dataset has the same protocol as we have in section 5.1, so we will not repeat the same experiment, and we compare gained results in table 5.5 in that section with our results in this section. We have trained our model once with the original training set of each dataset and once with the M3C dataset.

Table 5.8: Mean Confusion matrix for gender classification (Adience) by M3C

Actual \ Predicted	Male	Female	Recall
Male	**0.9069**	0.0985	0.8887
Female	0.0931	**0.9015**	0.9178
Precision	0.9069	0.9015	-
Specificity	0.9178	0.8887	-
F-Score	0.8977	0.9095	-

Table 5.9: Mean Confusion matrix for gender classification (MORPH-II) by M3C

Actual \ Predicted	Male	Female	Recall
Male	**0.9580**	0.1260	0.9115
Female	0.0420	**0.8740**	0.9389
Precision	0.9581	0.8740	-
Specificity	0.9389	0.9115	-
F-Score	0.9342	0.9052	-

Table 5.10: Overall accuracy for gender classification on benchmark datasets

Author \ Dataset	LFW	Adience	MORPH-II
Our model trained by original training set	-	89.2%	91.47%
Our model trained by M3C dataset	**98.19%**	**90.4%**	**92.24%**
CNN + Fine tuning + oversampling (van de Wolfshaar et al., 2015)	-	87.2%	-
CNN (Levi, G., & Hassner, T. 2015).	-	86.8%	-
LBP + FPLBP + SVM (Eidinger et al., 2014)	-	76.1%	-
LBP + SVM (Ramón-Balmaseda, 2012)	75.10%	-	88%
LBP + Adaboost (Shan, 2010)	94.44%	-	-
Boosted LBP + Adaboost (Shan, 2012)	94.81%	-	-
BIF + LPQ + BSIF (Gupta, Kumar, Yadav, & Shrivastava, 2018)	96.7%	-	97.1%

We have computed mean confusion matrices and evaluation metrics for Adience, MORPH-II and LFW datasets which are illustrated in tables 5.8, 5.9 and 5.5 respectively. All these confusion matrices, show a very good classification accuracy. As it is illustrated in 5.10, in both approaches (with and without M3C dataset) that we have employed to train our model, we have gained highest accuracy comparing to other methods on same datasets as we have used.

5.2.2 Age Group Classification

We have used the Adience dataset to evaluate our model for gender classification. Since the protocol is to use 5-fold cross-validation, we use four folds as the training set and the rest as the test or validation set of the Adience dataset for our experiment. As we explained in the previous chapter, we report the average value of the results for five different runs using 5-fold cross-validation and also the results for the 1-off method. The overall accuracy results for exact and 1-off methods are illustrated in table 5.11. The confusion matrix,

precision, recall, specificity and f1 score for age classification over Adience dataset are in the tables 5.12 and 5.13.

Table 5.11: Mean age classification accuracy and 1-off accuracy results on Adience

Method / Author	Exact age group	1-off age group
Our model	**76.44 ± 0.1**	**97.62 ± 0.6**
LBP+FPLBP (Eidinger et al., 2014)	44.5 ± 2.3	80.7 ± 1.1
LBP+FPLBP+Dropout 0.8 (Eidinger et al., 2014)	45.1 ± 2.6	79.5 ± 1.4
CNN with single crop (Levi & Hassner, 2015)	49.5 ± 4.4	84.6 ± 1.7
CNN with oversampling (Levi & Hassner, 2015)	50.7 ± 5.1	84.7 ± 2.2
DEX w IMDB-WIKI (Rothe, 2018)	64.0 ± 4.2	96.6 ± 0.9
DEX w/o IMDB-WIKI (Rothe, 2018)	55.6 ± 6.1	89.7 ± 1.8
DCNN (J.-C. Chen et al., 2016)	52.88 ± 6	88.45 ± 2.2

Table 5.12: Mean confusion matrix for age classification on Adience (Exact)

	0-2	4-6	8-13	15-20	25-32	38-43	48-53	60+	Recall
0-2	**0.760**	0.106	0	0	0	0.003	0	0	0.887
4-6	0.201	**0.703**	0.079	0.026	0	0.003	0	0	0.678
8-13	0.019	0.175	**0.829**	0.110	0.010	0.003	0.008	0	0.744
15-20	0.003	0.009	0.091	**0.726**	0.043	0.028	0.008	0	0.689
25-32	0.008	0.003	0	0.11	**0.872**	0.125	0.017	0.008	0.888
38-43	0.005	0.003	0	0.004	0.069	**0.706**	0.325	0.008	0.715
48-53	0	0	0	0	0.004	0.128	**0.583**	0.416	0.427
60+	0.003	0	0	0.018	0	0.003	0.058	**0.567**	0.839
Precision	0.760	0.703	0.829	0.726	0.872	0.706	0.583	0.567	-
Specificity	0.960	0.956	0.972	0.972	0.953	0.956	0.978	0.978	-
F-Score	0.819	0.690	0.784	0.707	0.880	0.711	0.493	0.677	-

Table 5.13: Mean confusion matrix for age classification on Adience (1-off)

	0-2	4-6	8-13	15-20	25-32	38-43	48-53	60+	Recall
0-2	**0.961**	0	0	0	0	0.003	0	0	0.997
4-6	0	**0.984**	0	0.026	0	0.003	0	0	0.978
8-13	0.019	0	**1**	0	0.010	0.003	0.008	0	0.955
15-20	0.003	0.009	0	**0.951**	0	0.028	0.008	0	0.939
25-32	0.008	0.003	0	0	**0.985**	0	0.017	0.008	0.989
38-43	0.005	0.003	0	0.004	0	**0.959**	0	0.008	0.984
48-53	0	0	0	0	0.004	0	**0.967**	0	0.975
60+	0.003	0	0	0.018	0	0.003	0	**0.983**	0.952
Precision	0.961	0.984	1	0.951	0.985	0.959	0.967	0.983	-
Specificity	0.993	0.998	1	0.995	0.994	0.994	0.998	0.999	-
F-Score	0.979	0.981	0.977	0.945	0.987	0.971	0.971	0.967	-

5.2.3 Age Estimation

As we explained in the previous chapter, we train our model with the M-Ages dataset then we perform test following the respective protocol for each dataset that we mentioned before in the previous chapter. In table 5.14, we have illustrated the results for FG-NET, LAP, CACD, and MORPH-II datasets. Regarding the FG-NET and LAP datasets, we have achieved better MAE compared to other methods in table 5.14 and regarding the CACD and MORPH-II datasets we have achieved much lower MAE compared to other methods. As it is illustrated in table 5.15, we have achieved smaller ϵ-error for LAP dataset.

Table 5.14: Average MAE (years) for age estimation on benchmark datasets

Author \ Dataset	FG-NET	LAP	CACD	MORPH-II
Our model	**3.02**	**3.22**	**4.28**	**2.21**
LBP Kernel Density Estimate (Ylioinas, 2013)	5.09	-	-	-
Label Distribution (CPNN) (Geng, Yin, & Zhou, 2013)	4.76	-	-	-
Biologically InspiredAAM (Hong, Wen, Fang, & Ding, 2013)	4.18	-	-	-
DEX (Rothe, 2018)	4.63	5.580	6.521	3.25
DEX with IMDB-WIKI (Rothe, 2018)	3.09	3.252	-	2.68
Method by (Guo & Mu, 2010)	-	-	-	4.46
Pretrained CNN model, softmax (Tan, Zhou, Wan, Lei, & Li, 2016)	-	-	-	3.03
BIF+LR (Liu, Lei, Wan, & Li, 2015)	-	-	7.79	-
BIF+SVR (Guo & Mu, 2013)	-	-	7.67	-
Fused method (Wan, Tan, Lei, Guo, & Li, 2018)	-	-	5.24	2.95

Table 5.15: Average $\epsilon-$error for age estimation on LAP dataset

Author \ Dataset	LAP
Our model	**0.276**
DEX (Rothe, 2018)	0.469
DEX with IMDB-WIKI (Rothe, 2018)	0.282

5.3 H-HP-ELM vs Softmax

We have performed two independent experiments for Softmax and H-HP-ELM classifiers, as we explained in chapter 3 to classify outputs from autoencoders of our proposed architecture. The idea of this experiment is to compare the performance of the classification layer only. In table 5.16 our results for H-HP-ELM and results for conventional architectures is illustrated in table 5.17. Model evaluation for the LFW dataset is trained using M3C, and it is evaluated by the testing set of the LFW dataset. Regarding the age classification, we have performed this experiment for 3 age categories, as we explained earlier.

Table 5.16: Accuracy and Training time for H-HP-ELM

	Age Classification		Gender Classification	
Dataset	Accuracy	Time	Accuracy	Time
MORPH-II	87.30%	01:12:51	91.47%	00:59:37
WIKI	85.14%	00:05:03	83.42%	00:05:10
IMDB	84.50%	00:28:33	82.20%	00:31:21
LFW	-	-	98.19%	-
ADIENCE	93.57%	00:38:22	89.20%	00:36:54
M3C	93.91%	01:47:19	94.03%	01:31:41

Table 5.17: Accuracy and Training time for Fully Connected Layers + Softmax

	Age Classification		Gender Classification	
Dataset	Accuracy	Time	Accuracy	Time
MORPH-II	83.95%	02:49:03	78.20%	02:03:44
WIKI	86.35%	00:22:34	81.95%	00:31:01
IMDB	79.65%	02:13:41	75.35%	02:24:00
LFW	-	-	98.34%	-
ADIENCE	79.90%	02:39:28	81.45%	02:44:13
M3C	87.23%	04:01:51	90.08%	03:17:21

As it is illustrated, we have analyzed age and gender classification in terms of training time and overall accuracy for each dataset. The fact that both models are identical in feature extraction layers but with different classifiers, we can then compare the impact of H-HP-ELM in terms of accuracy and training time. For example, results for the MORPH-II dataset clearly, show an improvement in both accuracy and training time. The training time for the Softmax model is 2 hours, while for the H-HP-ELM, it is almost 1 hour, and the overall accuracy improved from 83.95% and 78.20% to 87.30% and 91.47% for age and gender classification respectively. Similar improvements happened in other datasets as it is shown in tables 5.16 and 5.17. This experiment proves that we have achieved our objective to overcome the limitation of Softmax, considering the achieved results.

5.4 Summary

In this chapter, we have illustrated our experimental results by metrics that we explained in the previous chapter. We have presented how the H-HP-ELM layer can improve classification problems and how it can overcome the Softmax limitation that we explained in the first chapter. We have shown how gender classification, age classification and age estimation can be improved by using H-ELM classifiers and regressors.

CHAPTER 6: CONCLUSION AND FURTHER WORKS

6.1 Conclusion

We have achieved a better solution for age and gender classification in terms of accuracy and performance based on the results from the previous section comparing to the methods that are using Softmax as the classifier. We have also achieved a better solution for age estimation compared to the methods that are using other regressors as the age estimator. We have fulfilled our research objectives successfully as explained below:

- We managed to build our architecture using Hierarchical Extreme Learning Machines as the classifier part. The algorithm 4 in chapter 3, illustrates how we used HP-ELM in Hierarchical form by using a sparse autoencoder (algorithm 1 in chapter 3) and high performance ELM to classify age and gender. In section 3 in chapter 5, we have shown how accuracy and training time are improved by using H-HP-ELM instead of Softmax. Considering the results from section 5.3, we have clearly met first and second objectives.

- In chapter 3 we explained how supervised and unsupervised features are extracted. We eliminated the classifier and fully connected layers of a pre-trained CNN model to feed-forward the input through the whole network to extract unsupervised features. We also used HOG and Action Units as our supervised feature extraction techniques. We used three independent autoencoders to compact and fuse these three feature groups. In chapter 5 our benchmark evaluation results illustrates that we have achieved better accuracy and MAE compared to other methods even with Softmax classifier. It shows feature extraction methods that we have used are successfully representing good features of face of each individual. In other words we have successfully fulfilled the third and forth objectives.

In this research, we also realized that using a pre-trained CNN model can help to avoid under/over-fitting if the dataset is too small. When we are using small datasets for CNN networks, it is most likely to over-fit or under-fit because convolutional layers can not generalize with small datasets, but in this research, because we have used a pre-trained CNN, we could avoid this problem.

6.2 Further Work

Our research still suffers from a couple of limitations that can be addressed by other researchers for further researches. The very first limitation is that we need a large amount of data for training. There are limited non-commercial labeled facial datasets for age and gender, which can be addressed by other researchers to establish research on it and to collect systematic and useful facial images from different individuals with different ages to have more diversity in facial image datasets. Another problem for deep networks that we can address is that we need a high amount of RAM and GPU, which, although there are few solutions for them still they are not the best solutions. ELM networks suffer from uncertainty issues that can be addressed for further research.

REFERENCES

Akusok, A., Björk, K.-M., Miche, Y., & Lendasse, A. (2015). High-Performance Extreme Learning Machines: a Complete Toolbox for Big-Data Applications. *IEEE Access*, *3*, 1011–1025.

Aleksander, I., De Gregorio, M., França, F. M. G., Lima, P. M. V., & Morton, H. (2009). A Brief Introduction to Weightless Neural Systems. In *Esann* (pp. 299–305).

Azzopardi, G., Greco, A., & Vento, M. (2016). Gender Recognition From Face Images Using A Fusion of SVM Classifiers. In *International Conference on Image Analysis and Recognition* (pp. 533–538).

Baltrušaitis, T., Mahmoud, M., & Robinson, P. (2015). Cross-Dataset Learning and Person-specific Normalisation for Automatic Action Unit Detection. In *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on* (Vol. 6, pp. 1–6).

Baltrušaitis, T., Robinson, P., & Morency, L.-P. (2016). Openface: an Open Source Facial Behavior Analysis Toolkit. In *Applications of Computer Vision (WACV), 2016 IEEE Winter Conference on* (pp. 1–10).

Beck, A., & Teboulle, M. (2009). A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal on Imaging Sciences*, *2*(1), 183–202.

Bengio, Y., Boulanger-Lewandowski, N., & Pascanu, R. (2013). Advances In Optimizing Recurrent Networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference On* (pp. 8624–8628).

Bengio, Y., et al. (2009). Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, *2*(1), 1–127.

Chen, B.-C., Chen, C.-S., & Hsu, W. H. (2014). Cross-Age Reference Coding for Age-invariant Face Recognition and Retrieval. In *Proceedings of the European Conference on Computer Vision (ECCV).*

Chen, J.-C., Kumar, A., Ranjan, R., Patel, V. M., Alavi, A., & Chellappa, R. (2016). A Cascaded Convolutional Neural Network for Age Estimation of Unconstrained Faces. In *2016 IEEE 8th International Conference on Biometrics Theory, Appli-*

cations and Systems (btas) (pp. 1–8).

Chen, Y., Jiang, H., Li, C., Jia, X., & Ghamisi, P. (2016). Deep Feature Extraction and Classification of Hyperspectral Images Based on Convolutional Neural Networks. *IEEE Transactions on Geoscience and Remote Sensing*, *54*(10), 6232–6251.

Dahl, G. E., Sainath, T. N., & Hinton, G. E. (2013). Improving Deep Neural Networks for Lvcsr Using Rectified Linear Units and Dropout. In *Acoustics, Speech and Signal Processing (icassp), 2013 IEEE International Conference on* (pp. 8609–8613).

Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. In *Computer Vision and Pattern Recognition, 2005. Cvpr 2005. IEEE Computer Society Conference on* (Vol. 1, pp. 886–893).

Damelin, S. B., & Miller Jr, W. (2012). *The Mathematics of Signal Processing* (No. 48). Cambridge University Press.

Eidinger, E., Enbar, R., & Hassner, T. (2014). Age and Gender Estimation of Unfiltered Faces. *IEEE Transactions on Information Forensics and Security*, *9*(12), 2170–2179.

Escalera, S., Fabian, J., Pardo, P., Baro, X., Gonzalez, J., Escalante, H. J., ... Guyon, I. (2015). Chalearn Looking at People 2015: Apparent Age and Cultural Event Recognition Datasets and Results. In *Proceedings of the IEEE International Conference on Computer Vision Workshops* (pp. 1–9).

Fabian Benitez-Quiroz, C., Srinivasan, R., & Martinez, A. M. (2016). Emotionet: An Accurate, Real-time Algorithm for The Automatic Annotation of a Million Facial Expressions in the Wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5562–5570).

Felzenszwalb, P., McAllester, D., & Ramanan, D. (2008). A Discriminatively Trained, Multiscale, Deformable Part Model. In *Computer Vision and Pattern Recognition, 2008. cvpr 2008. IEEE Conference on* (pp. 1–8).

Geng, X., Yin, C., & Zhou, Z.-H. (2013). Facial Age Estimation by Learning From Label Distributions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(10), 2401–2412.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. (http://www.deeplearningbook.org)

Guo, G., & Mu, G. (2010). Human Age Estimation: What Is The Influence Across Race and Gender? In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops* (pp. 71–78).

Guo, G., & Mu, G. (2013). Joint Estimation of Age, Gender and Ethnicity: CCA vs. PLS. In *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)* (pp. 1–6).

Gupta, R., Kumar, S., Yadav, P., & Shrivastava, S. (2018). Identification of Age, Gender, & Race SMT (Scare, Marks, Tattoos) from Unconstrained Facial Images Using Statistical Techniques. In *2018 International Conference on Smart Computing and Electronic Enterprise (icscee)* (pp. 1–8).

Hess, U., Adams Jr, R. B., & Kleck, R. E. (2004). Facial Appearance, Gender, and Emotion Expression. *Emotion*, *4*(4), 378.

Hinton, G. (2010). A Practical Guide to Training Restricted Boltzmann Machines. *Momentum*, *9*(1), 926.

Hong, L., Wen, D., Fang, C., & Ding, X. (2013). A new Biologically Inspired Active Appearance Model for Face Age Estimation by Using Local Ordinal Ranking. In *Proceedings of the Fifth International Conference on Internet Multimedia Computing and Service* (pp. 327–330).

Huang, G. B., Ramesh, M., Berg, T., & Learned-Miller, E. (2007, October). *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments* (Tech. Rep. No. 07-49). University of Massachusetts, Amherst.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme Learning Machine: Theory and Applications. *Neurocomputing*, *70*(1), 489–501.

Huang, H.-Y. (2009). *Gender Differences In Facial Expressions of Emotions* (Unpublished doctoral dissertation). Humboldt State University.

Huerta, I., Fernández, C., Segura, C., Hernando, J., & Prati, A. (2015). A Deep Analysis on Age Estimation. *Pattern Recognition Letters*, *68*, 239–249.

Ivakhnenko, A. G. (1971). Polynomial Theory of Complex Systems. *IEEE Transactions On Systems, Man, and Cybernetics*, *1*(4), 364–378.

Javidan Darugar, M., & Loo, C. K. (2017). Gender Estimation Based on Supervised Hog, Action Units and Unsupervised cnn Feature Extraction. In *Artificial Intelligence and Robotics (IRANOPEN), 2017* (pp. 23–27).

Lanitis, A., & Cootes, T. (2002). FG-NET Aging Data Base. *Cyprus College*, *2*(3), 5.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based Learning Applied To Document Recognition. *Proceedings of The IEEE*, *86*(11), 2278–2324.

Levi, G., & Hassner, T. (2015). Age and Gender Classification Using Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (pp. 34–42).

Liu, T., Lei, Z., Wan, J., & Li, S. Z. (2015). DFDNET: Discriminant Face Descriptor Network for Facial Age Estimation. In *Chinese Conference on Biometric Recognition* (pp. 649–658).

Ma, J., Lu, C., Zhang, W., & Tang, Y. (2014). 1255. Health Assessment and Fault Diagnosis for Centrifugal Pumps Using Softmax Regression. *Journal of Vibroengineering*, *16*(3).

Nazir, M., Jan, Z., & Sajjad, M. (2018). Facial Expression Recognition using Histogram of Oriented Gradients Based Transformed Features. *Cluster Computing*, *21*(1), 539–548.

Nigam, S., Singh, R., & Misra, A. (2018). Efficient Facial Expression Recognition using Histogram of Oriented Gradients in Wavelet Domain. *Multimedia Tools and Applications*, *77*(21), 28725–28747.

Ollivier, Y., Tallec, C., & Charpiat, G. (2015). Training Recurrent Networks Online Without Backtracking. *arXiv preprint arXiv:1507.07680*.

Press, W. H. (1989). *Numerical Recipes In Pascal: The Art of Scientific Computing* (Vol. 1). Cambridge University Press.

Ramón-Balmaseda. (2012). Gender Classification in Large Databases. In *Iberoamerican Congress on Pattern Recognition* (pp. 74–81).

Ricanek, K., & Tesafaye, T. (2006). Morph: A Longitudinal Image Database of Normal Adult Age-progression. In *Automatic Face and Gesture Recognition, 2006. fgr 2006. 7th International Conference on* (pp. 341–345).

Robinson, A. J. (1994). An Application of Recurrent Nets to Phone Probability Estimation. *IEEE Transactions on Neural Networks*, *5*(2), 298–305.

Rothe. (2018). Deep Expectation of Real and Apparent Age From a Single Image Without Facial Landmarks. *International Journal of Computer Vision*, *126*(2-4), 144–157.

Rothe, R., Timofte, R., & Gool, L. V. (2015, December). Dex: Deep Expectation of Apparent Age From a Single Image. In *IEEE International Conference on Computer Vision Workshops (iccvw).*

Rothe, R., Timofte, R., & Gool, L. V. (2016, July). Deep Expectation of Real and Apparent Age From a Single Image Without Facial Landmarks. *International Journal of Computer Vision (IJCV)*.

Ruder, S. (2016). An Overview of Gradient Descent Optimization Algorithms. *arXiv preprint arXiv:1609.04747*.

Sainath, T. N., Mohamed, A.-r., Kingsbury, B., & Ramabhadran, B. (2013). Deep Convolutional Neural Networks for lvcsr. In *Acoustics, Speech and Signal Processing (icassp), 2013 IEEE International Conference On* (pp. 8614–8618).

Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, *61*, 85–117.

Shan, C. (2010). Gender Classification on Real-life Faces. In *International Conference on Advanced Concepts for Intelligent Vision Systems* (pp. 323–331).

Shan, C. (2012). Learning Local Binary Patterns for Gender Classification on Real-world Face Images. *Pattern Recognition Letters*, *33*(4), 431–437.

Sun, Y., Chen, Y., Wang, X., & Tang, X. (2014). Deep Learning Face Representation by

Joint Identification-verification. In *Advances in Neural Information Processing Systems* (pp. 1988–1996).

Szegedy, C., Toshev, A., & Erhan, D. (2013). Deep Neural Networks for Object Detection. In *Advances in Neural Information Processing Systems* (pp. 2553–2561).

Tan, Z., Zhou, S., Wan, J., Lei, Z., & Li, S. Z. (2016). Age Estimation Based on a Single Network With Soft Softmax of Aging Modeling. In *Asian Conference on Computer Vision* (pp. 203–216).

Tang, J., Deng, C., & Huang, G.-B. (2016). Extreme Learning Machine for Multilayer Perceptron. *IEEE Transactions on Neural Networks and Learning Systems*, *27*(4), 809–821.

Valstar, M. F., Jiang, B., Mehu, M., Pantic, M., & Scherer, K. (2011). The First Facial Expression Recognition and Analysis Challenge. In *Automatic Face & Gesture Recognition and Workshops (fg 2011), 2011 IEEE International Conference on* (pp. 921–926).

van de Wolfshaar, J., Karaaba, M. F., & Wiering, M. A. (2015). Deep Convolutional Neural Networks and Support Vector Machines for Gender Recognition. In *2015 IEEE Symposium Series on Computational Intelligence* (pp. 188–195).

van Heeswijk, M., & Miche, Y. (2015). Binary/ternary Extreme Learning Machines. *Neurocomputing*, *149*, 187–197.

Verma, S., & Jariwala, K. N. (2018). Age & Gender Classification using Histogram of Oriented Gradients and Back Propagation Neural Network.

Wan, J., Tan, Z., Lei, Z., Guo, G., & Li, S. Z. (2018). Auxiliary Demographic Information Assisted Age Estimation With Cascaded Structure. *IEEE Transactions on Cybernetics*, *48*(9), 2531–2541.

Wang, W., Huang, Y., Wang, Y., & Wang, L. (2014). Generalized Autoencoder: A Neural Network Framework for Dimensionality Reduction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops* (pp. 490–497).

Wang, Y., Yao, H., & Zhao, S. (2016). Auto-encoder Based Dimensionality Reduction.

Neurocomputing, *184*, 232–242.

Widrow, B., Greenblatt, A., Kim, Y., & Park, D. (2013). The No-Prop Algorithm: A New Learning Algorithm for Multilayer Neural Networks. *Neural Networks*, *37*, 182–188.

Wu, W., Qian, C., Yang, S., Wang, Q., Cai, Y., & Zhou, Q. (2018). Look At Boundary: A Boundary-Aware Face Alignment Algorithm. In *Cvpr*.

Ylioinas. (2013). Age Estimation Using Local Binary Pattern Kernel Density Estimate. In *International Conference on Image Analysis and Processing* (pp. 141–150).

Zadeh, A., Chong Lim, Y., Baltrusaitis, T., & Morency, L.-P. (2017). Convolutional Experts Constrained Local Model for 3d Facial Landmark Detection. In *Proceedings of The IEEE International Conference on Computer Vision* (pp. 2519–2528).

Zhao, W., & Du, S. (2016). Spectral–spatial Feature Extraction for Hyperspectral Image Classification: A Dimension Reduction and Deep Learning Approach. *IEEE Transactions on Geoscience and Remote Sensing*, *54*(8), 4544–4554.

Zhou, B., Koerger, H., Wirth, M., Zwick, C., Martindale, C., Cruz, H., ... Lukowicz, P. (2016). Smart Soccer Shoe: Monitoring Foot-ball Interaction With Shoe Integrated Textile Pressure Sensor Matrix. In *Proceedings of the 2016 ACM International Symposium on Wearable Computers* (pp. 64–71).

Zhu, Q., Yeh, M.-C., Cheng, K.-T., & Avidan, S. (2006). Fast Human Detection using a Cascade of Histograms of Oriented Gradients. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* (Vol. 2, pp. 1491–1498).