COMBINING DEEP LEARNING WITH TRADITIONAL MACHINE LEARNING TO IMPROVE CLASSIFICATION ACCURACY ON SMALL DATASETS

by

Ghadeer Ahmed H Alabandi, B.S.

A thesis submitted to the Graduate Council of
Texas State University in partial fulfillment
of the requirements for the degree of
Master of Science
with a Major in Computer Science
December 2017

Committee Members:

Vangelis Metsis, Chair

Byron Gao

Habil Zare

COPYRIGHT

by

Ghadeer Ahmed H Alabandi

2017

FAIR USE AND AUTHOR'S PERMISSION STATEMENT

Fair Use

This work is protected by the Copyright Laws of the United States (Public Law 94-553, section 107). Consistent with fair use as defined in the Copyright Laws, brief quotations from this material are allowed with proper acknowledgement. Use of this material for financial gain without the author's express written permission is not allowed.

Duplication Permission

As the copyright holder of this work I Ghadeer Ahmed H Alabandi, authorize duplication of this work, in whole or in part, for educational or scholarly purposes only.

DEDICATION

Dedicated this thesis to my parents and my sister for their continued love and support. I also dedicated this work to my husband and my son who have always been with me throughout my journey and never stop supporting me

ACKNOWLEDGEMENTS

I want to thank my thesis advisor Dr. Vangelis Metsis for his guidance and support throughout my thesis which kept me motivated to do the best job I can do. I also want to thank my committee Dr. Byron Gao and Dr. Habil Zare, who have given their time, so I that can successfully complete my thesis.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF ABBREVIATIONS

CNNs - Convolutional Neural Network.

EEG - Electroencephalogram

ECG - Electrocardiogram

EMG - Electromyography

EOG - Electrooculography

EDA - Electrodermal Activity

Accel - Accelerometer

Gyro - Gyroscope

ReLU - Rectified Linear Units

Conv - Convolutional

FLC - Fully Connected Layer

SVM - Support Vector Machine

LG - Logistic Regression

RF - Random Forest

DT - Decision Tree

KNN - k-Nearest Neighbors

SA - Standard Accuracy

MAA - Macro Average Accuracy

ABSTRACT

Feature extraction and selection are essential phases in building machine learning classification models, and they have a great impact on the accuracy and the performance of the model. However, these phases are expensive, and there is no guarantee that manually extracted features will generalize well in different data modalities. Deep learning models integrate the phases of feature extraction, selection, and classification into a single optimization process. However, they are very computationally expensive compared to traditional machine learning algorithms, and they require large training datasets to achieve good classification performance.

This work explores ways of combining the advantages of deep learning and traditional machine learning models by building a hybrid classification scheme. The first few layers of a convolutional neural network are utilized for feature extraction and selection. Subsequently, the extracted features are fed to a traditional supervised learning algorithm to perform classification. We evaluate our method on sensor data coming from human physiological biosignal measurements and motion tracking data coming from accelerometers. Our experimental results show that our hybrid approach outperforms deep learning and traditional machine learning algorithms when those are used in isolation on small da

1. INTRODUCTION

Biosignals have become a significant indicator for medical and psychological diagnosis. Extracting meaningful features from these signals is very important to understanding human functional state and diagnosing any harmful disease accurately. Biosignals are signals collected using sensors and used in diagnosis of diseases and Physiological therapy [11]. However, the analysis of biosignals requires detecting meaningful events of interest in the signals. Each biosignal has different characteristics and features, e.g. amplitude and frequency [2]. The majority of the research involving biosignals focuses on hand-engineering features, which require human experts to design algorithms to extract meaningful features from biosignals for each specific application. For example, meaningful features extracted from EEG signals for sleep staging are different from the features that we may need to extract from EEG signals for emotion recognition. This makes feature extraction and selection a complicated task. The goal of this research is to develop a method that uses deep learning to detect events of interest, from any type of biosignal after processing biosignals in an appropriate way to give us a high accuracy. Feature extraction is an essential phase in building a trusted model, especially in the medical and physiological field where patients are involved, which make the model accuracy highly important. Furthermore, due to regulations and privacy concerns, the number of available health-related datasets are limited both in number and in size. Different machine learning algorithms have been used for making decisions based on the data given to the model. However, the accuracy of these algorithms relies heavily on the features extracted from the raw data. There is a variety of feature selection methods being used, but they are usually computationally expensive and there is no guarantee that they will

converge to the optimal features. For example, in [3] the accuracy of the SVM algorithm varies depending on the feature selection method used. In feature-dependent methods, the main difficulty is to extract the appropriate features. In certain types of data, to extract high quality features we need human-like understanding of the raw data. Deep learning comes to solve this problem by eliminating the need for separate feature extraction, selection and model training phases. Deep learning has shown significant improvement in image classification and object detection. In early object detection approaches, people extracted features and fed these features to learning algorithms (e.g. SVM) to successfully detect objects of interest (e.g. pedestrians) in the image. However, when these methods were used to detect several classes other than pedestrians e.g. car, sign or tracks, the accuracy of the model dropped [4]. The use of deep convolutional neural network showed a notable increase in the performance of detecting objects using highly challenging datasets [5]. In addition, deep learning has been used to generate audio-visual features for emotion recognition [6]. One of the most important research problems is to develop a method to extract features from biosignals that can be efficient for that task. In the last few years, deep learning has become one of the most significant ways to extracting features from raw data which make feature extraction less dependent on human experts. These research works demonstrate promising results that deep learning can learn to select the most appropriate features from the raw data and can allow automation feature extraction from physiological raw data.

1.1 Motivation

Biosignals have been widely used in a variety of applications for medical diagnosis, psychological data analysis and other health-related applications, as shown in Table 1 and Table 2. In most of these applications the types of features to be extracted from the signals were manually specified. In these applications biosignals go through different steps before feeding them to the classifier. Firstly, signals pre-processed to ensure that only signals with good quality can pass to the next phase. Secondly, time-domain, frequency-domain and time-frequency-domain features are extracted from each recording. The features are then passed to feature selection algorithm to select only irredundant and significant features. The selected features are then used to train the classifier to identify the right label for each instance. However, these features extracting methods may not generalize well in larger populations due to the diversity between subjects and recording environment. This problem occurs because of the hand-engineering approach of extracting features from biosignals, based on their characteristics in the available dataset. Hand-engineering tends to be used by a variety of biosignal-related applications to reduce the amount of data fed to the classifier and to improve the generalization of the model.

Deep learning is a branch of machine learning which consists of linear and nonlinear multilayer processing neurons. A deep neural network utilizes these layers and neurons to extract features from raw data. In images and audio, the features extracted from the deep learning are very effective and outperform other machine learning classifier that have been trained using feature extracted using hand-engineering methods. However, deep learning has not been yet successfully utilized for biosignals analysis, mainly due the limited number of datasets available for the task. In this work, we develop a hybrid learning model that can detect various events of interest, from different physiological signals, while

eliminating the need for separate feature extraction and selection steps. We test this model on different classification problems that involve physiological signals. One of the main drawbacks of deep learning is that it requires large amounts of data to successfully train multiple layers of neurons and provide high classification accuracy. To overcome the need for large amounts of data, we will experiment with hybrid learning methods, where at the lowest level, only a few layers of a convolutional neural network are used to automatically extract features from raw biosignals. Subsequently, the extracted features from different layers will be fed to classic supervised learning algorithms (e.g. SVM) which require smaller amounts of data to achieve high accuracy.

1.2 Biosignals and their Applications

Table 1 lists a set of biosignals, their originating tissue/organ and the physiological characteristic that they measure. Table 2 list a set of applications where biosignals have been used along with corresponding published work.

Table 1: Electrical Biosignals, Biosignals can be also none-electrical including acoustic, mechanical, magnetic, optic and chemical signals.

Biosignal	Tissue/Organ	Measure
Electroencephalogram (EEG)	Brain	Brain Activity
Electrocardiogram (ECG)	Heart	Heart Rate
Electromyography (EMG)	Muscles	Muscles Activity
Electrooculography (EOG)	Eye	Eye Movement
Electrodermal Activity (EDA)	Skin	Skin Conductance

Table 2: Biosignals application examples.

	Applications	**Signals**
[7]	Sleep analysis	EEG
[8]	Emotion recognition	EMG, ECG, EDA and other non-electrical biosignals.
[9]	Lie detection	EEG, EDA
[10]	Stress Monitoring	ECG, EDA
[11]	Seizures detection	EEG, ECG
[12]	Brain-Computer Interfaces for Speech Communication.	EEG, EMG

1.3 Challenges

The key challenge in this research is optimizing the Convolutional Neural Network (CNN) for each data set. There are a number of aspects that need to be considered during building and optimizing the CNNs. To fully optimize the CNNs we should be aware that the value of hyperparameters can affect how smooth is the learning process and the model behavior. The CNNs hyperparameters are the number of layers, the size of the filters for convolution and the convolution stride for each convolutional layer, the pooling region size and the pooling stride for each pooling layer, and the number of units for each fully-connected layer. The number of layers, the size and the number of features map in each layer can significantly change the deep model behavior and the features that the model can learn as shown in figure 1. Large number of layers can cause overfitting while small number of layers can cause underfitting. The size of the filters can also affect the learning process because different size of filters captures different features and usually large filters capture less features than the small filters. Another aspect that we need to be aware of is the

regularization penalty value. The L2 weight decay lambda can limit the model learning if the value chosen is too large or too small, however without the L2 weight decay the model can easily overfit to the noise in the signals which result in undesired behavior because this weight decay helps the model to learn smoother filters. Also setting the learning rate without suffering from Knockout Problem because we are using the ReLU activation function and this can easily be avoided by applying batch normalization. The need of optimizing and fine tuning the hyperparameters means we must run the CNNs model unspecific number of time and we are aware that that the CNNs training process takes a significant amount of time.

1.4 Summary of Methodology and Findings

In this research, we introduce a hybrid model that can learn features from the raw singles. The main purpose of this model is to automate the process of features extraction by utilizing the feature extraction capabilities of deep learning. In our model, we utilize Convolutional Neural Networks (CNNs) to extract features, and then we fed the extracted features to a traditional supervised learning algorithm to perform the classification. We evaluate our model on sensor data coming from human physiological biosignal measurements and motion tracking data coming from accelerometers. We compared the performance of our model with the previous studies that have been conducted on the same datasets that we used, where they utilize hand-engineering features for classification. The results demonstrated that our model can achieve a similar performance compared to the state-of-the-art methods, and without utilizing any hand-engineered features. We believe that our approach provides a general framework for classifying sensor signals.

2. BACKGROUND AND RELATED WORK

2.1 Biosignals Processing

The traditional approach for biosignal analysis consist of several phases that require a human expert in the loop. The first phase is pre-processing the signals to ensure only quality signals is remain. This pre-processing includes correcting inaccurate signals, removing aircrafts, normalizing the signals to a range of values, and passing them through filters to remove any noise.

The second phase is extracting features from the pre-processed signals. The extracting method depends heavily on a human expert who can specify which features are meaningful and useful for the intended application. The Fourier transform is one of the most common tools to extract frequency domain features. Another tool that is commonly used to extract features from biosignals is the Wavelet transforms which can extract time domain features. The third phase is to select a subset from the extracted features before passing the features to the machine learning model. This phase reduces the number of features by selecting the most significant features, to reduce redundancy and speed up the training process. It also helps the model to generalize well without overfitting. The last phase is to pass the selected features to the machine learning classifier to build a trained model [13].

2.2 Using Deep learning on Biosignals

There are some works that have used deep learning for biosignals but most of the published research applies deep neural networks for specific applications, which uses deep learning to extract features and predict the classes labels. On this section, we will discuss a few early examples of work that use deep learning with physiological signals.

Using deep learning for automatic sleep staging based on the EEG signals, Tsinalis et.al. applied a Convolutional Neural Network (CNN) for sleep staging. They used CNN

to extract features from a single channel EEG signal and to classify the sleep stages. They used raw EEG signals and without any signal pre-processing. By feeding raw data to CNNs they were able to compare the result with a research that they conducted on the same dataset using hand-engineering time-frequency analysis-based feature extraction and Morlet wavelet. Their model is effective in extracting features from a single channel EEG signal for sleep staging application only. They did not use polysomnogram (PSG) signals and their model has been tested only on EEG signals that have been collected from the Fpz-Cz channel [14]. Martinez, et.al. used deep learning to extract features from biosignals for emotion recognition. They studied and analyzed the impact of using different feature extraction and selection methods compared to the feature extracted from feeding raw signals to the Convolutional Neural Network (CNN). They used deep learning network for feature extraction and for classification. They used EMG, EDA and blood volume pulse signals that have been collected for emotion recognition purpose. They conclude that using deep learning for automatic feature extraction could be as effective as ad-hoc manual feature extraction methods. They also expect deep learning to be more effective when using large physiological datasets. [15]. Stober et.al., at the Brain and Mind Institute at University of Western Ontario, applied a CNN to detect events of interests from EEG signals which have been collected from different channels to determine what music the subject is listening to during the EEG recording. For better accuracy, all stimuli from all subjects has been normalized to the same length and the EEG and EOG signals have been pre-processed to remove unwanted artifacts. The main purpose of their work is to extract interpretable features from EEG that can be used to distinguish between the different music stimuli [16]. Ian Walker, at the Imperial College of Science Technology and Medicine, developed a

deep learning model to classify user intent generated by motor imagery and EEG signal. The main purpose of his research is to analyze brainwaves in order to convert user intent into actionable signals. He applied CNNs to EEG signals generated from motor imagery to classify user intent. He used this model to develop a pilot game to help pilots get through the obstacles that they may face in the real-world training course by using a virtual training course [17].

3. METHODOLOGY

The building methodology for our hybrid model consist of two main phases: 1) build and train a CNN using early stopping to stop the training if no improvement in the accuracy is obtained after a certain number of steps; 2) extract features the convolutional and fully connected layers and pass them to traditional machine learning classifiers.

3.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are biologically-inspired variants of the Multiple Layer Perceptrons (MLPs) [18]. They are special kind of neural networks and the most widely applied deep learning model in last few years. They have gained momentum in the field of computer vision, especially in two-dimensional image applications. In the medical field, they have been mostly used in medical imaging, e.g. Brain MRI, for detecting tumors. However, CNNs have not been widely used in biosignals applications, where one-dimensional biosignals, such as electroencephalography (EEG), are utilized. Recently there have been some efforts which applied CNNs in biosignal related applications, especially in sleep staging applications that use EEG signal channels. Even though CNNs have been introduced as a type of neural network they have a special architecture that make them stand over the ordinary neural networks. The architecture of CNNs usually is composed of a convolutional layer, activation function, and a max pooling layer, also known as down-sampling layer. The convolutional layer implements a convolution operation. CNNs are built based on three basic ideas, i.e., local receptive fields, weight sharing, and pooling [19].

3.1.1 Local Connectivity and Parameters Sharing

Local connectivity, or sparse connectivity, cuts down the computational cost without affecting the performance. Unlike the fully connected layers in neural networks where all neurons in one layer are connected to all neurons in the next layer, the Convolutional Neural Networks (CNNs) are locally connected. Convolutional layers are technically locally connected layers with shared weight. This connectivity comes from passing the same filter over all unit positions, i.e. receptive field, on the input stream as shown in figure 1. These units share the same weight vector and bias. Sharing parameters in this way allows the CNNs to detect the features regardless of their position in the input stream. Moreover, parameter sharing improves the CNNs learning efficiency because it significantly reduces the number of the parameters being learnt and optimized, and enable the CNNs to achieve better generalization on the given problems [20].

3.2 Convolutional Neural Network Architecture

3.2.1 Convolutional Layer

Each Convolutional layer consists of a number of parameters for its filter. Usually the filter is small in comparison to the input volume, but these filters extend through the full depth of the input volume. In convolutional network terminology, the first argument the convolutional function takes is often referred to as the input, and the second argument is the kernel size while the output is the feature map that is going to be the input for the next layer. In machine learning applications, the input is usually a multidimensional array of data and the kernel of the filter is usually a multidimensional array of parameters that are adapted by the learning algorithm. These multidimensional arrays are known as tensors because each element of the input and kernel must be explicitly stored separately. For

example, if a convolutional layer has an input with size 5x5x20 this size can be explained as the Height x Width and the number of filters in the layer. It is worth mentioning that this format may change from one platform to another, i.e. in TensorFlow [21] the format of convolutional layer is height x width x number of filters in the layer while in Theano [22] the format is different. However, in both Theano and TensorFlow the forward propagation goes through the same process of sliding each filter across the width and height of the input volume in order to compute the dot products that result from adapting the filter over the input tensor. As a result of sliding the filter over the width and height of the input volume the output of the layer will be a 2-dimensional activation map and this map is the input for the next layer. During the training of forward and backward propagation, the filters will learn, with the help of the activation function. At each convolutional layer, the filter will learn different features, and as we go further in the CNNs, the filters in the deeper layers will learn complex patterns of features. The resulting feature map from applying the filter in each layer will be the stacked along the depth dimension and produce the output volume for the next layer. Moreover, in each convolutional layer we need to set the value of the stride. The value of the stride controls how the filter is applied to the input and thus controls the number of feature maps resulting from applying this filter to the input. Stride refers to the spacing of the receptive fields in the previous layer. Figure 1 presents a convolutional layer with a stride of two and filter of size 3x3 over an input stream of size 5 x 5. Each Element in the output grid is computed by elementwise multiplying the input with the filter and summing it up [23].

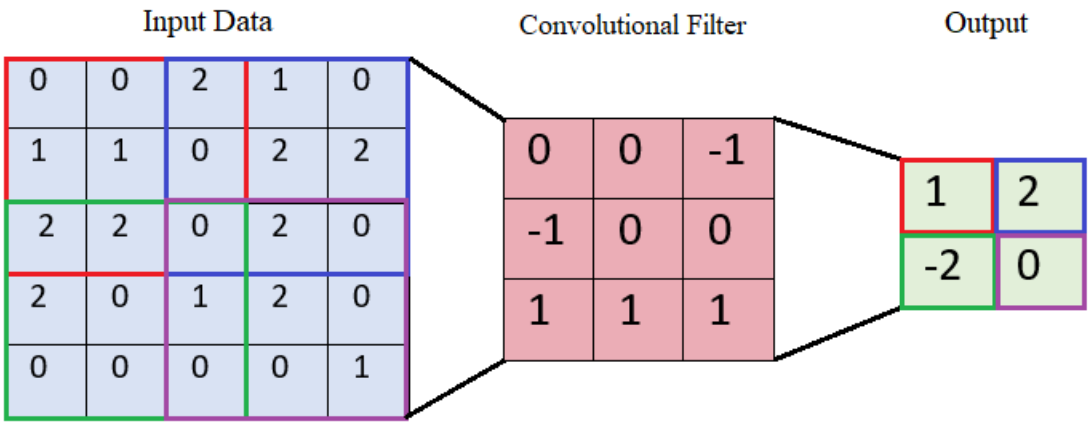

Figure 1. Example of a learned 3 × 3 convolutional filters passing over a 5 × 5 input with stride = 2 to produce a 2 × 2 output. In the input grid the unite positions or receptive fields are marked with red, blue, green and purple are sharing the same filter weights.

3.2.2 Max Pooling Layer

It is common to place a pooling layer after the convolutional layers in most CNN applications. The main function of the pooling layer is to reduce the spatial size of the input by reducing the number of the parameters and in result reducing the computation size in CNNs. Pooling layers also control the overfitting problem that learning applications face usually when using the deep learning models. The pooling layer operates independently on every depth slice of the input and resizes that slice by using the max operation. The max pooling function reports the maximum output within a rectangle neighborhood. This means the pooling layer helps to make the representation invariant to small transitions of the input. Small invariant transition means that the value of the output does not change as we translate the input by the same factor. What this means is that in most CNNs application, like face recognition, the goal is to recognize if there is a face in the image, not the pixel position of the face in the image. So, by shifting or down sampling the size of the image we still have

a face on the image even though the pixel position or the size of the image is different than the original input. An example of max pooling is if we used 2 by 2 pooling size with stride size of 2 which is the most common stride size with max pooling, the pooling layer will down sample every input depth slice by 2 [24]. The MAX operation would in this case be taking a max over 4 numbers, i.e. square region of 2 by 2 as shown in figure 2.

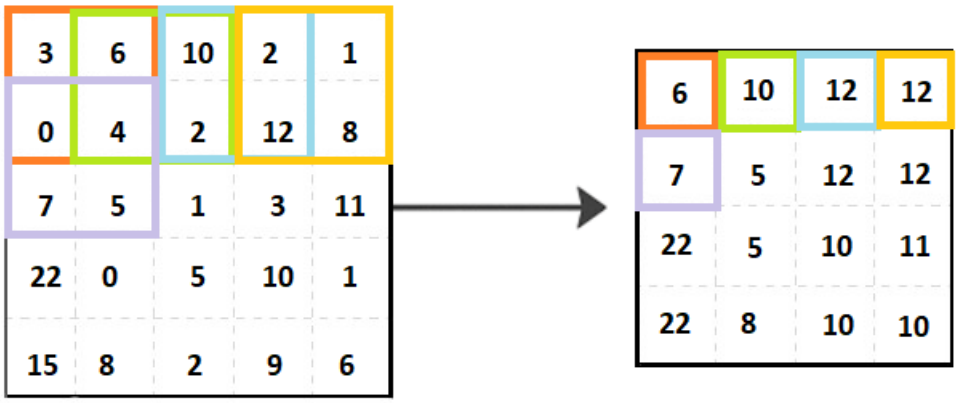

Figure 2. Max pooling with 2 x 2 pooling size and stride size 1.

3.2.3 Fully Connected layer:

The output from the last convolutional layer, or last max pooling layer, is fed to fully connected layer as an input. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

3.3 Convolutional Neural Network Optimization

3.3.1 Batch Normalization

A recently developed technique by Ioffe and Szegedy [25] called batch normalization improves the deep neural network initialization by explicitly forcing the activations functions throughout a network to take on a unit Gaussian distribution at the beginning of the training. The main reason of using batch normalization is to avoid killing the activation function and the gradient vanishing problems. A large gradient could cause the weight to be updated in a way that the neuron will never be activated on any data point again. The gradient vanishing problem could happen if a change on the deep net parameters causes a very small change in the output which means the network is not able to learn effectively. Using batch normalization helps to avoid these problem that could affect the performance of the neural network. This technique works by normalizing activations throughout the network to prevent any small changes to the parameters from having a larger change in the gradients convergence. This means it prevent the change in learning rate from causing gradient non-convergence or gradient vanishing. In neural networks, if the parameter is too small or too large this may affect the gradient during the backpropagation and either causes model exploding or model non-convergence. By using batch normalization, the gradient during the backpropagation will not be affected by the scale of the learning rate. On the other hand, the batch normalization is not guaranteed to normalize the values to be Gaussian nor independent and can't guarantee to help the gradient propagation to behave better during the training process [25].

3.3.2 Regularization

There are several ways to prevent the Neural Networks from overfitting, especially when the training dataset is not large

L2 Regularization

L2 is a regularization technique with a weight decay parameter, which adds a penalty into the loss function to prevent large values of the parameters in the model. The weight decay is called lambda and this parameter needs to be fine-tuned because it can limit the model capabilities of learning long-term dependencies. The lambda value decides the degree of penalty that we want to add to the loss function and can affect how smoother the CNN filters are. L2 is the most common form of regularization in CNNs [26].

L1 Regularization

The L1 regularization has the property that makes the weight vectors to become sparse during optimization. Neurons with L1 regularization use only a sparse subset of the input which makes these neurons able to recognize the noise in the input. However, L1 takes long computation time compared to L2 and this is because final weight vector that results from using L2 is usually diffuse while using L1 the final weight vectors are large sparse vectors. In most cases L1 regularization is used to help perform feature selection in sparse feature spaces. In practice, it has been proved that L2 preforms than L1 [27].

Dropout Layer

Dropout is an extremely effective, simple and recently introduced regularization technique by Srivastava et al [28] that complements the other methods (L1, L2). The dropout layer works by temporarily dropping out some random units in the neural network. The network drops out these units along with all their incoming and outgoing connections. In the training phase, dropout is implemented by only keeping a neuron active with some probability (a

hyperparameter) while in the testing phase, the drop out probability will be set to zero. The output at test time is the same as the output at training time even though at the test time the removed units are present, but the weights of these units are multiplied by the dropout probability. In this work, we employee dropout layers and we fine tune the dropout probability to find the best value. The number of dropout layers and their location is important because it can affect the performance significantly.

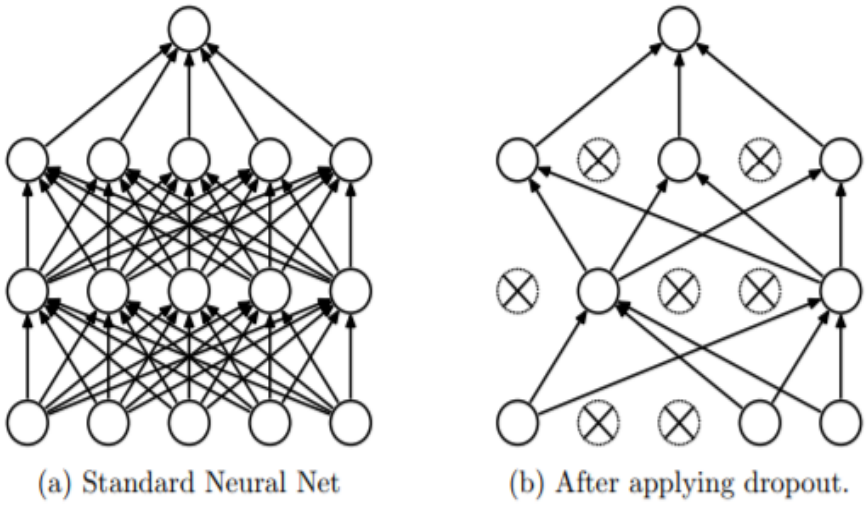

Figure 3. It shows the difference between the Fully Connected Neural Network before and after adding Dropout Layers [28].

3.3.3 Activation Function (Rectified linear unit (ReLU))

Activations are a major choice point when designing neural networks. They determine how the inputs are transformed throughout the network, which is essential for the network's ability to learn complex functions. A purely linear activation function could be chosen at each layer, but then the outputs would be simply linear transformations of the inputs, thus ruling out the ability to learn more complex functional forms. Thus, non-linear activations are preferred for their increased flexibility.

Traditionally, researchers have used the sigmoidal (1.0) function and tangent (1.1) in the past because they are both non-liner and have easily computable derivatives, which is significant for efficient computation of the optimization function.

$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad (1.0)$$

$$\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \quad (1.1)$$

In all experiments, we used the ReLU activation function (1.2) because It has been proved to be computationally faster than other common activation function (i.e. tanh and sigmoid) and because it doesn't need any normalization or expensive operations such as exponential computation [29]. It also has been proved by Krizhevsky et al that ReLU is six times faster than tanh on CIFAR-10 [30]. It also does not suffer from the gradient vanish problem. However, using ReLU may cause another problem. If the learning rate used within the optimization function is large the CNNs may suffer from the Knockout Problem. This problem occurs when the large gradient causes the weights of some neuron to be updated in such way that the neuron activation function, i.e. ReLU, will be fragile and will never activated again at any data point and the gradient of this neuron will be zero through the training process. However, to prevent this problem we apply batch normalization when the learning rate used is large. The ReLU works by using the MAX operation:

$$ReLU(z) = \max(0, z) \ (1.2)$$

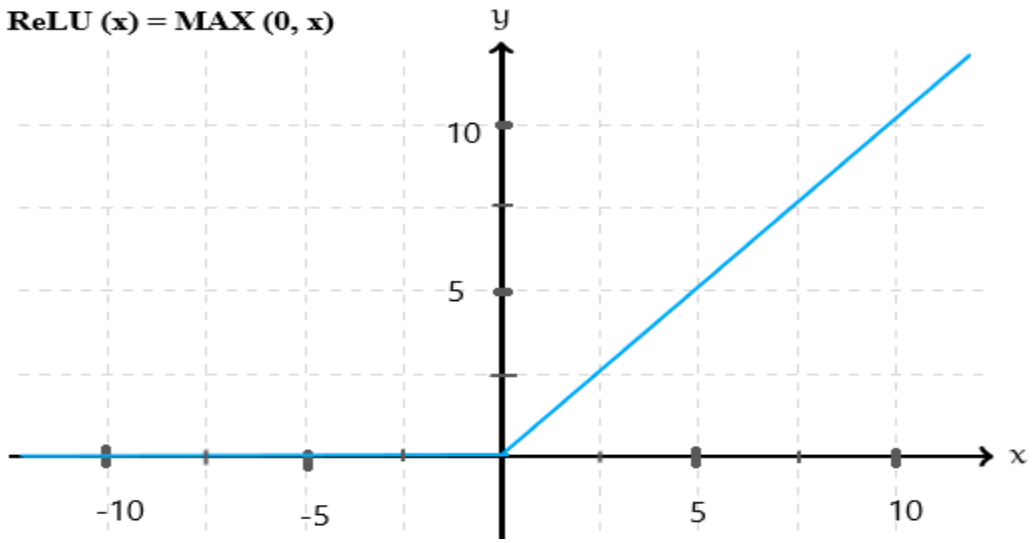

Figure 4. The blue line represents how the ReLU activation function work.

3.3.4 Loss Function (SoftMax with Cross-Entropy)

This loss function computes the SoftMax and Cross-Entropy between the real label and the predicted labels. The combination of these two cost functions measures the probability of the error in the discrete classification tasks where classes are mutually exclusive. For example, each data segment is labeled with only one label and no two labels are assign to the same data segment or instance.

SoftMax Classifier

It is a type of activation layer that is usually used for multi-classification in logistic regression models. It interprets the outputs of the fully connected layer or the last layer in the deep learning network as probabilities. The calculated probabilities are in range between 0 and 1 and the sum of these probabilities is equal to 1. Mathematically the SoftMax function is shown below:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

Where z is a vector of the inputs to the output layer, j is set of the output units indexes, j = 1, 2, ..., K [31].

Cross Entropy Cost Function

If a function that the deep learning use to measure the error at a SoftMax layer.

$$C = -\frac{1}{n} \sum_n [yIn\ a + (1-y)In(1-a)]$$

Where n is the total number training inputs, the sum is over all training inputs, x, and y is the corresponding desired output [31].

3.3.5 Mini Batch (Batch size)

The batch term means group of examples, and in deep learning it is the number of data instances being used in each forward and backward pass cycle to optimize the deep model. Batch size is the term that is usually used to describe the minibatch size. Batch size is important because it could limit the model capabilities of learning. Larger batches provide a more accurate estimate of the gradient. On the other hand, training with a batch size of one is not recommended because small batches could add noise to the learning process and may also cause generalization and sampling error. However, for some models where they have to use batch size equal to one or batch size less than 128 they might require using small learning rate to maintain stability. Using a very small learning rate will cause the deep learning model to take longer time to converge and more time to examine the whole training set. It is also crucial that the minibatches are selected randomly form the training data where the same batch could contain data from different subjects or data with different pattern to allow the model to generalize. Another crucial parameter is the number of Epochs because the number of Epochs multiplied by the batch size determine the number

of the forward and backward pass cycles. In each cycle, the optimization function will try to optimize the weight vector at each layer to reduce the loss, i.e. cost [32].

3.3.6 Optimization Algorithm (Adam optimization)

The main purpose of the optimization function is to find the parameters θ of a neural network that reduces the cost function J(θ) by significantly amount. J(θ) in our research is the SoftMax cross entropy cost function. This process includes an intensive performance measure to evaluate the entire training set. In the case of deep learning, we optimize functions that may have many local minima that are not optimal, and many saddle points surrounded by very flat regions. The optimization function in the deep neural network is considered as non-convex optimization function where there is only one global optimum for the optimization problem that optimization function trying to solve which is minimizing the average training loss. All of this makes optimization difficult, especially when the input to the function is multidimensional because of this increase the number of local minimums. We therefore usually settle for finding a value of that is very low but not necessarily the minimal. However, a problem may occur if all the local minimums have a high cost compared to the global minimum cost. Using gradient descent to optimize the neural network is not efficient because gradient descent can easily be stuck in the local minimum or due to having plateaus gradient descent may not converge. Stochastic gradient descent will converge if the learning rate is not constant throughout the optimization process which means to effectively use the stochastic gradient decent we need to apply the learning rate decay to guarantee convergence. Also, to improve the stochastic gradient descent optimization mini-batch is being used in the deep neural network where instead of using the gradient over single example the deep neural network will take the gradient of the

average regularized loss for all the mini-batch examples. This help in generalizing the deep net during the training process and because we are taking the gradient over multiple data example the gradient risk will be more accurate. Another technique called momentum which has been introduced by Polyak in 1964, help in improving the optimization results. Momentum computes the gradient by taking the exponential average of the previously computed gradient average. In this research we used the Adam optimization function, Adam is one of the adaptive learning rate optimization algorithm that updates the learning rate for different parameters from estimates of the first and second gradients moments. Adam optimizer is a combination of RMSProp [33] and momentum optimization but with a few important improvements that make Adam stand out. First, Adam adapt the parameters learning rats based on the average of the first and second moment. While in RMSProp the parameters learning rates are adopted based on the gradient first moment only, i.e. the average of the recent magnitudes of the gradient. Second, Adam includes bias corrections to for both the first-order moments and second-order moments. While RMSProp incorporates bias corrections for the first-order moment initialization only. For this reason, RMSProp may have a high bias in the second-order moment estimates. Specifically, Adam optimizer have additional parameters beta1 and beta2 to control the decay rates of the calculated exponential average move and the squared move of the gradient [34] [35]. These improvements give significant advantages to Adam over the other optimizer. On the original paper it has been proved that in CNNs Adam make a rapid progress lowering the cost of the training process and it converge faster than other functions [34].

3. 4 Convolutional Neural Network Representation Learning

We fully train CNNs with early stopping using different filter sizes to extract features from raw signals. In our model, we apply CNNs where each layer consists of max pooling and we apply dropout layers and batch normalization through the training process only. In the feature extraction phase and the testing process, we don't apply dropout layers. Each convolutional layer performs four operations sequentially: 1D-convolution with its filters, Max-Pooling, batch normalization, and applying the rectified linear unit (ReLU) activation (i.e., ReLU(x) = MAX (0, x)). Each pooling layer down samples inputs using a max operation. The model specification, such as the filter sizes, the number of filters, stride sizes and pooling sizes may vary depending on the characteristics of the dataset. However, the main idea is to extract features from these datasets using the best model specification. In each experiment that we run, we build a CNN with multiple convolutional layers with a small filter size. The use of multiple convolutional layers with a small filter size instead of a single convolutional layer with a large filter can reduce the number of parameters and the computational cost, and can still achieve the similar level of model expressiveness.

3. 5 Fine-Tuning

The second step after choosing the model specification is to perform a supervised fine-tuning on the whole model with a sequential training set. Then the model is trained with the sequence training set using a mini-batch and Adam optimizer with different learning rates. Fine tuning with different learning rates helps to prevent the model from overfitting. Also, we use different number of training epochs and we use the result to decide which parameter values are the best for the used dataset. Moreover, for each dataset we experiment with different convolutional neural network specifications, i.e. convolutional layer filter size, depth and width of the convolutional neural network, and the number of

hidden units in the fully connected layer. Surprisingly, the number of hidden units in the fully connected layer affect the performance of the CNNs as shown in figure 1, where different number of hidden layers where used to decide the most suitable number to be used in each experiment.

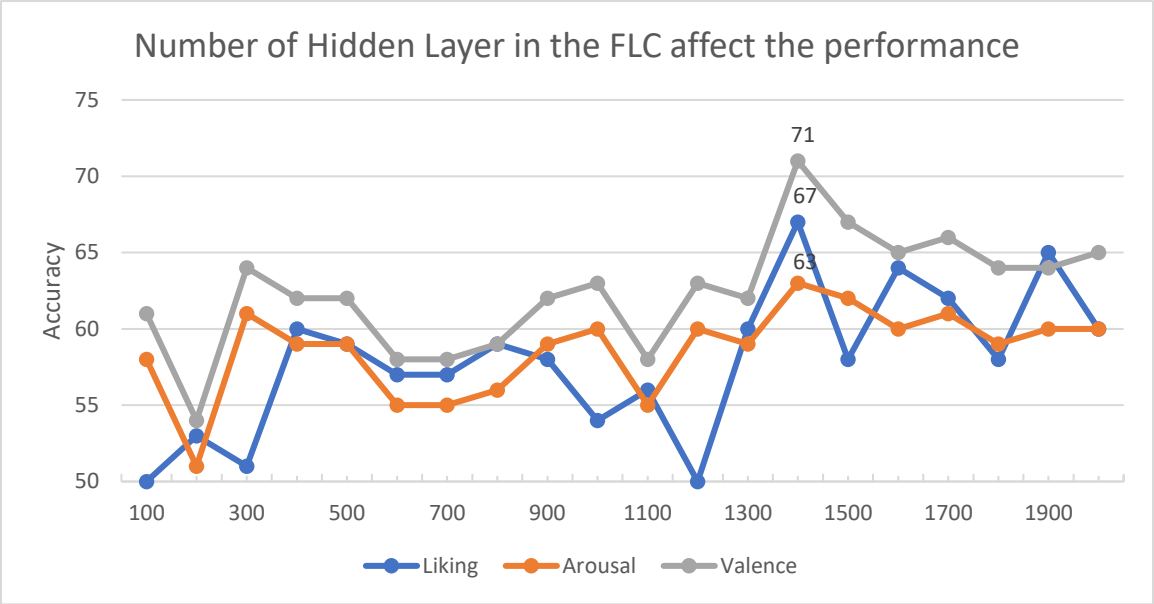

Figure 5. The effect of the Number of Hidden Layer (x axis) in the FLC on the CNNs accuracy.

3.6 Feature Extraction Process

After optimizing and training the CNNs, deep net features are extracted from different convolutional layers and from the fully connected layer. The features from a single layer are fed to the classic supervised machine learning algorithms.

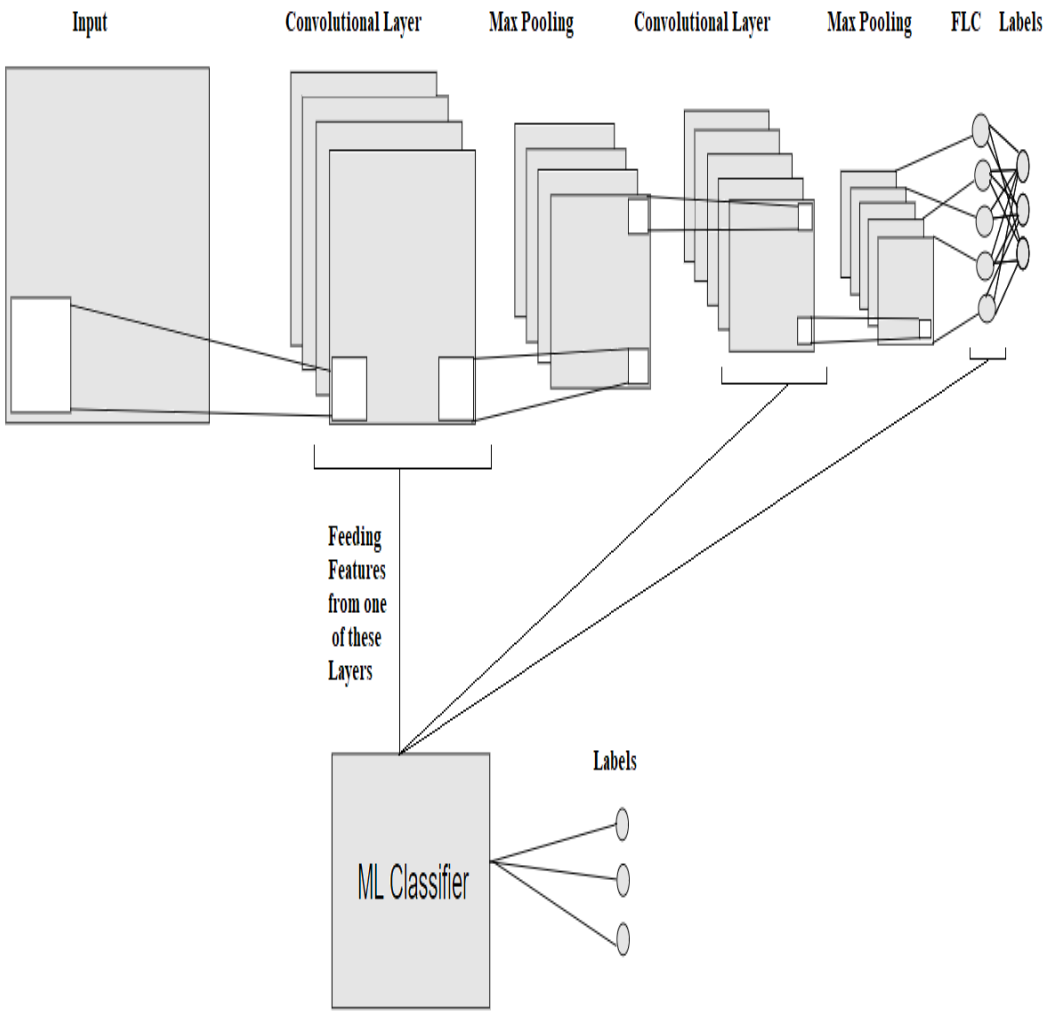

Figure 6. An overview of the hybrid model consisting of two main phases, fully trained CNNs and extracting and passing features to classical supervised machine learning algorithms.

3.6.1 Machine Learning Classifiers

We used five well-known machine learning classifiers to train and test with the features extracted from the CNN. Following we give brief definition of each classification method that we used:

Random Forest (RF)

Large numbers of decision trees are generated by using a random sampling of the data. These trees then vote for the most popular class.

K-Nearest Neighbor (KNN)

An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.

Support Vector Machines (SVM)

The space is separated by hyperplanes. The location of this hyperplanes depends on the training data. The algorithm outputs an optimal hyperplane which categorizes the new examples.

Decision tree (DT)

It is rule based classification method that generates a tree with the leaves representing the labels and the branches representing the conjunction of the features that can lead to different class labels.

Logistic Regression (LR)

It is a regression-based method of classifying data into discrete outcomes. It is usually used where the classification task is binary. In this work, we used multinomial logistic regression which generalizes the linear logistic regression and allows multiple discrete outcomes.

4. IMPLEMENTATION

We implemented our model using Google TensorFlow which is a deep learning library and we run the model using NVIDIA GeForce GT 740M.

4.1 TensorFlow

TensorFlow is the successor to DistBelief, which is the distributed system for training neural networks that Google has used since 2011. One of DistBelief limitations is the fact that DistBelief all deep net layers are implemented as C++ classes where most machine learning researchers familiar with python and R which limit the DistBelief range of researchers. Another limitation that google team was able to overcome in TensorFlow is having different optimization functions where DistBelief was only using stochastic gradient descent. Also, even after adding GPU support, DistBelief remains a heavyweight system for training deep neural networks on huge datasets, and is difficult to scale down to other environments. However, TensorFlow was designed to be much more flexible and not heavyweight while still meeting the demand of Google's production machine learning workloads. TensorFlow allows the user to implement applications on distributed clusters, local workstations, mobile devices, and custom-designed accelerators provides by using a simple dataflow-based programming abstraction. TensorFlow warp the construction of dataflow graphs by using high level scripting interface which allow the users to experiment with different model architectures and optimization algorithms without modifying the core system. A typical TensorFlow deep learning model goes through two distinct phases. The first phase is to define the dataflow graph with placeholders for input data and variables that represent the state like the weights and biases variables for each layer, while in the second phase the TensorFlow deep learning model will execute the optimized version of

the program. Moreover, Tensors in TensorFlow enable several optimizations for memory management and communication, such as RDMA and direct GPU-to-GPU transfer. TensorFlow use computational graphs because it enables auto-differentiation, i.e. it enables the forward pass computation to be extended automatically for computing the back-propagation using the chain rule. Using computational graphs can sometimes be inefficient and slow because it requires it to allocate memory and keep the graph during the training process. However, most deep learning models are complex and writing the backward pass manually is so complicated and fault prone. Also, TensorFlow has the benefit of being able to utilize multicore CPU and GPU setups, which improves the performance and saves power [36] [37].

5. EXPERIMENTS AND RESULTS

5.1 Applying Regularization

We employed two regularization techniques to help prevent overfitting problems. The first technique is dropout that randomly sets the input values to 0 (i.e., dropping units along with their connections) with the specified probability during training. Dropout layers with a probability of 0.5 were used throughout the model after convolutional layer and 0.7 after the fully connected layer. It is important to note that these dropout layers were used for training only, and were removed from the model during testing to provide deterministic outputs. The second technique is L2 weight decay, which adds a penalty term into a loss function to prevent large values of the parameters in the model. We apply this because without the weight decay, the filters in the CNNs overfitted to noise or artifacts, especially if the dataset is small. This weight decay helped the model learn more smoothly, which resulted in better performance gains. In our experiments, the value of the weight decay parameter that defines the degree of penalty, lambda, varies depending on the characteristic of the dataset.

5.2 Model Evaluation

To evaluate the effectiveness of our proposed hybrid model, we use sensor data coming from human physiological biosignal measurements and motion tracking data coming from accelerometers. The evaluation process consists of different phases. The first phase is to optimize the deep CNNs based on the dataset characteristic. The second phase is to fully train the deep CNNs. The third phase, that comes after training the CNNs with the best hyperparameters, is to extract features from the fully trained model and then feed these features to traditional machine learning classifiers. To accurately evaluate the deep

hybrid model's performance, we evaluate classification metrics on four raw datasets and we compare the results to the previous state-of-art approaches where feature extraction and selection methods were applied along with traditional machine learning classifiers. Moreover, in all the dataset we used a leave-one-subject-out cross validation to ensure that the hybrid model generalizes well in larger populations despite the diversity between subjects and recording environments.

5.3 Performance Metrics

We evaluated the performance using macro-averaging precision (PR), macro-averaging recall (RE), macro-averaging F1-score (F1), overall accuracy (ACC). Moreover, to overcome the class imbalance problem in the second experiment, we used two assessment metrics: the standard accuracy (SA) and the macro average accuracy (MAA).

5.4 The Model Architecture

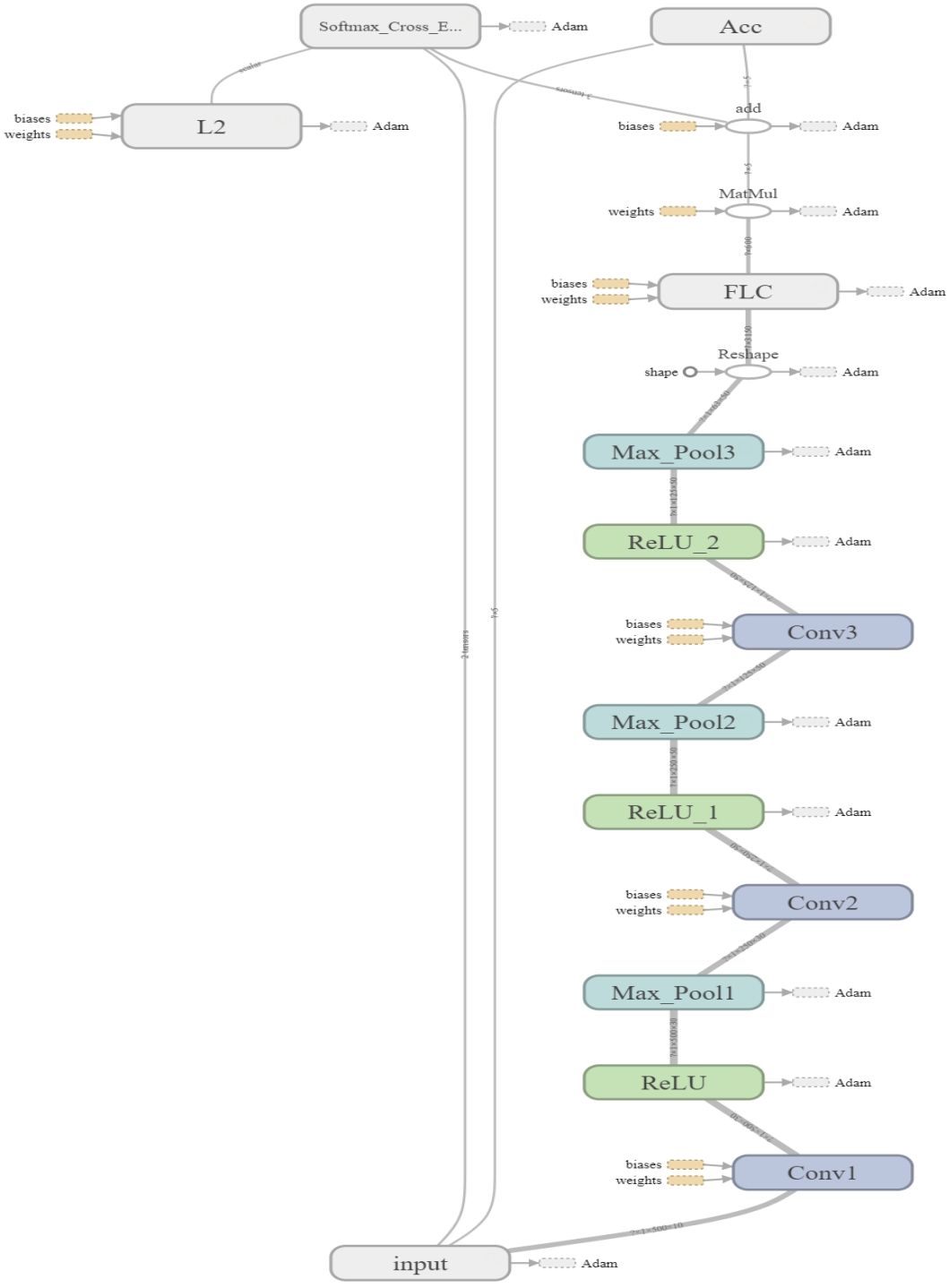

Figure 7. The deep CNNs Architecture for the first experiment. Summary of the selected hyperparameters for each layer are provided in table 3.

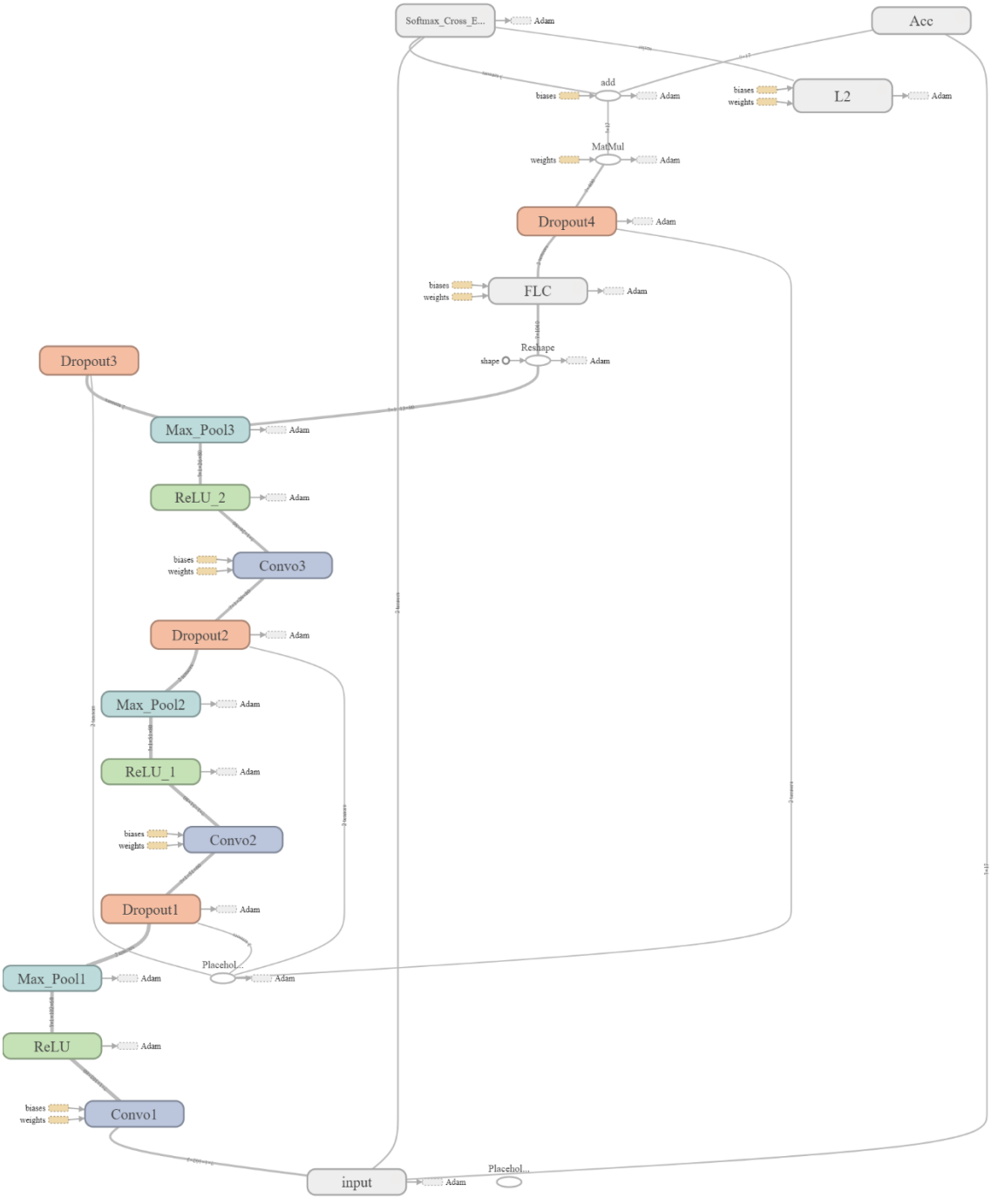

Figure 8. The deep CNNs Architecture for the second, third and fourth experiment. Summary of the selected hyperparameters for each experiment are provided in tables 7,11, 12, and 18.

5.5 Experiment one (Small Human activity dataset)

This dataset was collected at the IMICS Lab at Texas State University. This dataset consists of 3-axis Accelerometer, 3-axis Gyroscope, ECG and EMG signals. The data was collected using a Bio-Radio physiological monitoring device at a sampling rate of 250Hz and there was a total of approximately 40,000 data samples. The dataset consists of three subjects and each did five activities over a period of approximately 2 minutes and 40 seconds. The five activities that were performed by each subject are walking, sitting, standing, walking upstairs and downstairs. The signals were segmented using a fixed-width sliding window of 2 seconds with 50% overlap (500 samples/window).

5.5.1 Training Parameters

The model was trained with a batch of size of 194. The Adam optimizer's parameters lr, beta1, and beta2 were set to 0. 00001, 0.9 and 0.999 respectively. Then the whole model was fine-tuned using the sequential training set. Specifically, we equally split the sequences of 2-s epochs from each subject data into 194 sub-sequences (i.e., batch size). Then we fed 10 epochs from each sub-sequence yielding 1940 epochs per one step of training. With this setting, we found that the pre-training and fine-tuning steps started to converge eventually. It should be noted we used the default value for most optimizer parameters such as beta1, beta2. We experimented with the batch size (from 10 to 200) during the training, the epochs (from 5 to 100), the learning rates (from 10^{-1} to 10^{-6}), and l2 weight decay (from 10^{-1} to 10^{-6}). In this dataset, I did not use batch normalization and dropout layers because after using them the performance dropout.

5.5.2 Experiment one model specification

Table 3: The CNNs model specification for the small human activity dataset.

Conv layer 1	Filter: 6*6, Number of channels: 10, Number of filters: 30, Stride (1,1)
Max Pool 1	Kernel size (2, 2), Stride (2,2)
Conv layer 2	Filter: 5 * 5, Number of channels: 30, Number of filters: 50, Stride (2,2)
Max pool 2	Kernel size (2, 2), Stride (2,2)
Conv layer 3	Filter: 5 * 5, Number of channels: 50, Number of filters: 50, Stride (2,2)
Max pool 3	Kernel size (2, 2), Stride (2,2)
FLC	600

The approximate training time of CNNs with this dataset is 1 hours 29 seconds.

5.5.3 Experiment 1 Results

Table 4: The result from extracting features from the second convolutional layer.

Classifier	F1	Precision	Precision	Precision
CNN-RPF-SVM	74	75	74	75
CNN-KNN, k = 5	65	76	67	79
CNN-Logistic Regression	67	70	68	78
CNN- DT	65	66	65	73
CNN- RF, Trees = 20	73	85	72	83

Table 5: The result from extracting features from the third convolutional layer.

Classifier	F1	Precision	Recall	Accuracy
RPF-SVM	89	94	87	94
KNN, k = 5	76	90	76	85
CNN-Logistic Regression	73	85	71	82
CNN-DT	60	63	67	43
CNN-RF, Trees = 20	61	80	63	76

Table 6: The result from extracting features from the FCL.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM	80	85	80	81
CNN-KNN, k = 5	69	80	69	80
CNN-Logistic Regression	71	81	71	82
CNN-DT	62	65	63	69
CNN-RF, Trees = 20	73	85	70	83
CNN Only	84	85	84	85

5.5.4 The Confusion matrix for the highest accuracy

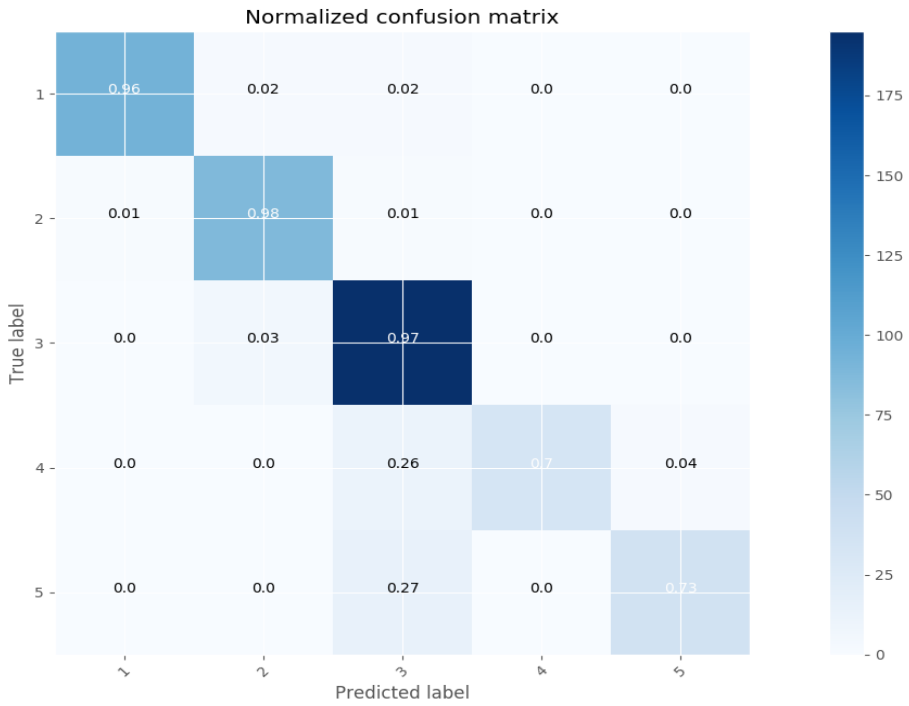

Figure 9. Confusion matrix of the classification using features map of the third convolutional layer with CNN-SVM.

5.5.5 Comparison with the State-of-the-Art Methods

A study was conducted using the same dataset with traditional machine learning approaches. The dataset processed by extracting the mean, root-mean-squared, and variance for all signals and the frequency domain and amplitudes for ECG and EMG. They examine the dataset based on the extracted features and they run the machine learning classifier with a combination of the features and the highest accuracy they score was 96 % using the data from Accelerometer and Gyroscope only. However, using data from all recorded signals they score 94.7 %. In our experiment we used the data from all the recorded signals and we score 95 % which is almost equal to the accuracy obtained with manual feature extraction. The highest accuracy that we score were from using the extracted feature from the third convolutional layer along with SVM with RPF kernel. Figure 10 shows a bar chart of the best accuracies obtained by the different feature selection and classification approaches.

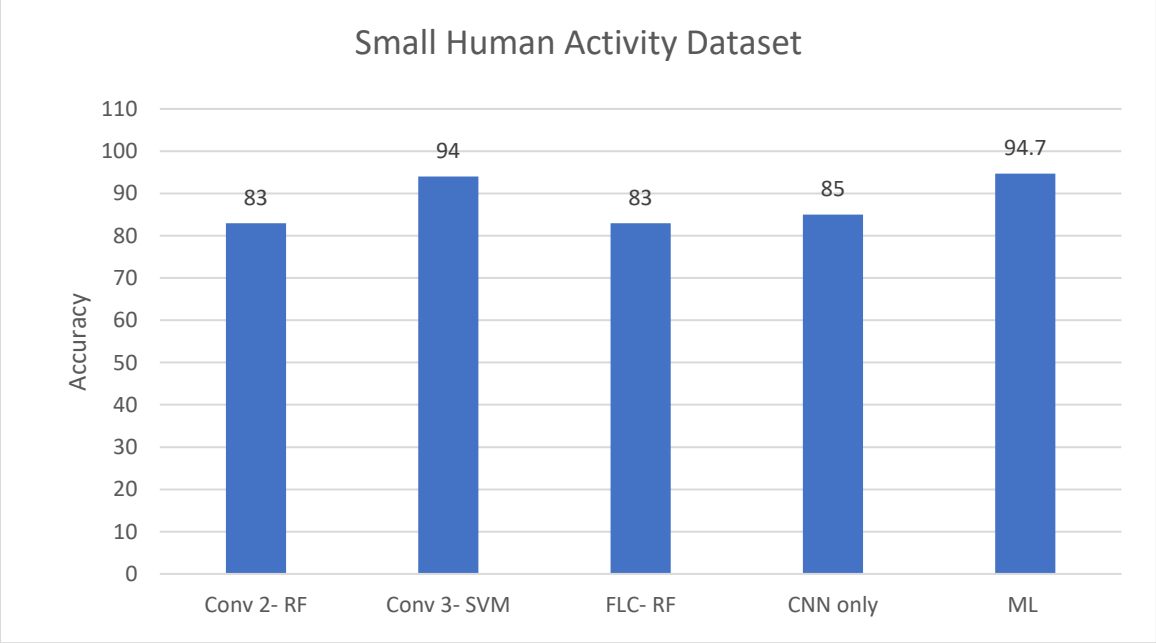

Figure 10. Summary of Experiment One Results.

5.6 Experiment two (Large human activity dataset)

The dataset called UNIMIB-SHAR consists of 17 classes including activities of daily living and fall events, and it contains a total of 7,013 recorded activities performed by 30 subjects of ages between 18 and 60 years. Activities are divided into 17 fine grained classes grouped in two coarse grained classes: 9 types of activities of daily living (ADL) and 8 types of falls. The data recorded through built-in triaxial accelerometer with a sample frequency of 50 Hz. In addition, the application records audio signals with a sample frequency of 8,000 Hz. This dataset consists of 3-axis Accelerometer (x, y, z) that represents the accelerations along each of the 3 Cartesian axes. The dataset divided into four subsets in this research we are using the largest subset AF-17 that contains all the 17 activity classes (standing up, getting up, walking, running, going up, jumping, going down, lying down, setting down, falling forward, falling right, falling back, falling obstacle, falling protection, falling chair, falling syncope, and falling left). [38].

5.6.1 Training Parameters

The model was trained with batches of size of 185. The Adam optimizer's parameters lr, beta1, and beta2 were set to 0. 0001, 0.9 and 0.999 respectively. Then the whole model was fine-tuned using the sequential training set. Specifically, we equally split the sequences of 1-s epochs from each subject data into 185sub-sequences (i.e., batch size). Then we fed 30 epochs from each sub-sequence yielding 5550 epochs per one step training. It should be noted we used the default value for most optimizer parameters such as beta1, beta2. We experimented with the batch size (from 10 to 200) during the training, the epochs (from 5 to 100), the learning rates (from 10^{-1} to 10^{-6}), and l2 weight decay (10^{-1} to 10^{-6}).

In this dataset, we use batch normalization and dropout layers with penalty 0.5 over all convolutional layers and 0.7 after the fully connected layer.

5.6.2 Experiment two model specification

Table 7: The CNNs model specification for the large human activity dataset.

Conv layer 1	Filter: 20*3, Number of channels: 3, Number of filters:120, Stride (1,1)
Max Pool 1	Kernel size (2, 2), Stride (2,2)
Conv layer 2	Filter:10*5, Number of channels: 120, Number of filters: 80, Stride (1,1)
Max pool 2	Kernel size (2, 2), Stride (2,2)
Conv layer 3	Filter: 10 * 5, Number of channels: 80, Number of filters: 80, Stride (2,2)
Max pool 3	Kernel size (2, 2), Stride (2,2)
FLC	400

The approximate training time of CNNs with this dataset is 2 hours 44 seconds.

5.6.3 Experiment two results

Table 8: The result from extracting features from the second convolutional layer.

Classifier	F1	Precision	Recall	Accuracy	SA	MAA
CNN- RPF-SVM	22	31	21	49	50.0	21.0
CNN-KNN, k = 2	45	52	44	59	60.0	44.76

Table 9: The result from extracting features from the third convolutional layer.

Classifier	F1	Precision	Recall	Accuracy	SA	MAA
CNN- RPF-SVM	65	65	67	71.0	71.14	57.5
CNN-KNN, k = 2	43	52	46	59	60.0	43.8

Table 10: The result from extracting features from the FLC layer.

Classifier	F1	Precision	Recall	Accuracy	SA	MAA
CNN- RPF-SVM	65	67	66	83	79.0	62.0
CNN-KNN, k = 2	67	69	68	84	80.68	66.82
CNN only	58	60	56	75	72.74,	56.17

5.6.4 The Confusion matrix for the highest accuracy

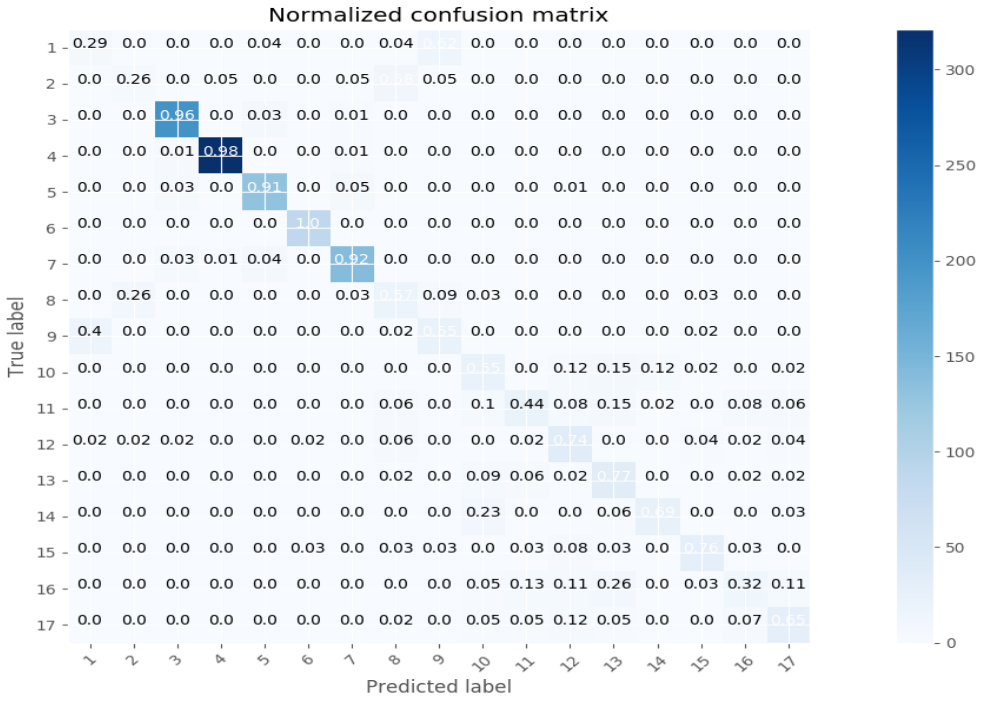

Figure 11. Confusion matrix of the classification using features of the FCL with CNN-KNN.

5.6.5 Comparison with the State-of-the-Art Methods

A study was conducted using the same dataset with traditional machine learning approaches. The dataset was processed by extracting the magnitude. They examine the dataset by basing the raw data and the magnitude feature vector to the machine learning classifier and they evaluate the performance using mean average accuracy (MAA) and standard accuracy (SA). The highest MAA and SA they score for AF-17 dataset are 58 and 51.14 using the magnitude feature vector [38]. In our experiment, the highest accuracy MAA and SA are 80.68 and 66.82. These results were from using the extracted features from the fully connected layer along with KNN. Comparing our model performance to the state-of-the-are approach [38] our model SA and MAA performances are better.

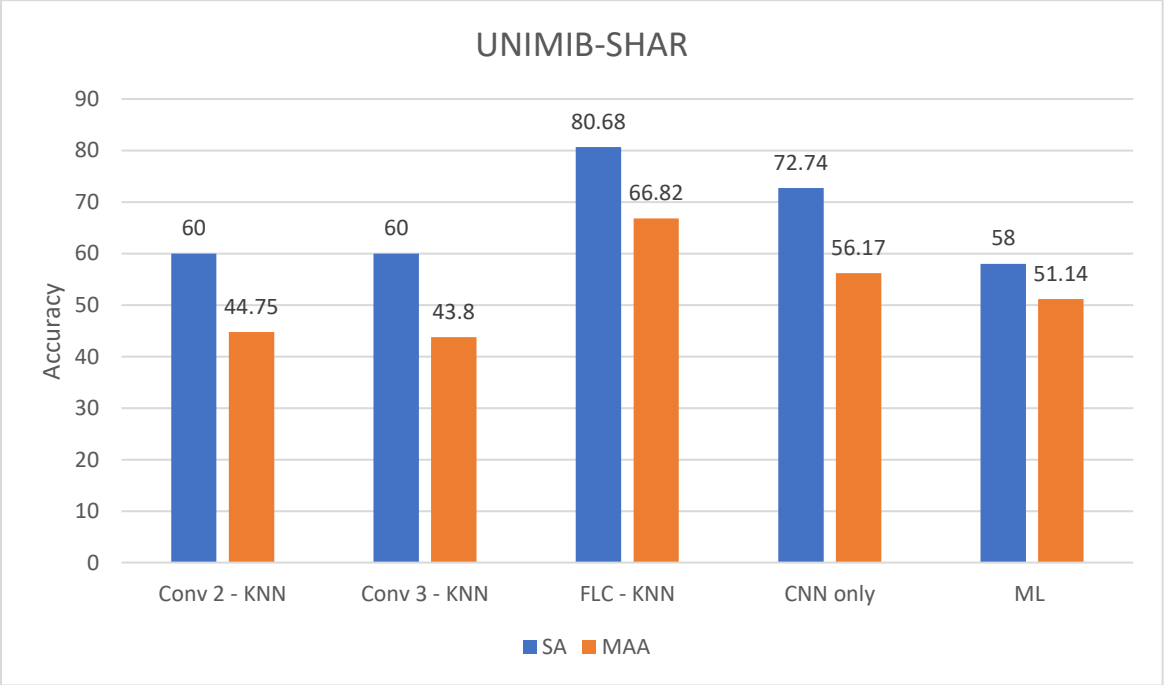

Figure 12. Summary of Experiment Two (UNIMIB-SHAR dataset) Results.

5.7 Experiment three (Small Emotion Dataset)

The dataset consists of data collected from 5 subjects at the IMICS Lab at Texas State University. The subjects were involved in seven Virtual Reality (VR) sessions including watching movies and playing games. The subjects self-reported the arousal level after each session. The arousal level used is low (-1), medium (0) and high (1). The classification labels are based on subjects self-reported and on further signals analysis from each session. The data were ollected using two BioRadio physiological monitoring devices. The data were recorded with a sample frequency of 250 Hz rate and they consist of these biosignals: EEG f4, EOG – Horizontal, EOG – Vertical, EMG – Zygomaticus "smile" muscle, Accel XYZ (Rear of head), Gyro XYZ (Rear of head), GSR (Electrodermal Activity), ECG, Chest Respiration (RIP), Abdomen Respiration (RIP), Peripheral Temperature, Heart Rate via PulseOx, Blood Volume (PPG) via PulseOx, Blood Oxygen (SpO2) via PulseOx, Accel XYZ (Right waist) and Gyro XYZ (Right waist) [39].

5.7.1 Training Parameters

The model was trained with batch of size of 122. The Adam optimizer's parameters lr, beta1, and beta2 were set to 0. 001, 0.9 and 0.999 respectively. Then the whole model was fine-tuned using the sequential training set. Specifically, we equally split the sequences of 1-s, 2-s and 10-s epochs from each subject data into 122 sub-sequences (i.e., batch size). Then we fed 10 epochs from each sub-sequence yielding 1220 epochs per one step training. It should be noted we used the default value for most optimizer parameters such as beta1, beta2. We experimented with the batch size (from 10 to 200) during the training, the epochs (from 5 to 100), the learning rates (from 10^{-1} to 10^{-6}), and l2 weight decay (10^{-1} to 10^{6}). In

this dataset, we use batch normalization and dropout layers with penalty 0.5 after last convolutional layers and 0.7 after the fully connected layer.

5.7.2 Experiment three model specification

Table 11: The model specification for 3 arousal levels (low, medium, high).

Conv layer 1	Filter: 6*6, Number of channels: 24, Number of filter: 120, Stride (2,2)
Max Pool 1	Kernel size (2, 2), Stride (4,4)
Conv layer 2	Filter: 5 * 5, Number of channels:120, Number of filter: 60, Stride (2,2)
Max pool 2	Kernel size (2, 2), Stride (4,4)
Conv layer 3	Filter: 5 * 5, Number of channels: 60, Number of filter: 30, Stride (2,2)
Max pool 3	Kernel size (2, 2), Stride (4,4)
FLC	600

Table 12: The model specification for binary arousal levels (low, high).

Conv layer 1	Filter: 6*6, Number of channels: 24, Number of filter: 70, Stride (4,4)
Max Pool 1	Kernel size (2, 2), Stride (4,4)
Conv layer 2	Filter: 5 * 5, Number of channels: 70, Number of filters: 60, Stride (4,4)
Max pool 2	Kernel size (2, 2), Stride (2,2)
Conv layer 3	Filter: 5 * 5, Number of channels: 60, Number of filters: 50, Stride (4,4)
Max pool 3	Kernel size (2, 2), Stride (2,2)
FLC	600

The approximate training time of CNNs with this dataset is 1 hours 20 seconds.

5.7.3 Experiment three results

Table 13: The result of extracting features from the third convolutional layer for arousal levels (low, medium, high).

Classifier	F1	Precision	Recall	Accuracy
CNN- RPF-SVM Gamma=0.01, C=10	90	89	92	90
CNN-KNN k = 5	67	79	65	72
CNN-DT Max Depth = 10	50	47	59	51
CNN-RF Trees = 5	75	76	77	75
CNN-logistic regression Penalty= 'l2', C= 30.5	87	92	84	90
CNN Only	85	86	85	82

Table 14: The result of extracting features from the third convolutional layer for binary arousal levels (low, high).

Classifier	F1	Precision	Recall	Accuracy
CNN- RPF-SVM Gamma=0.01, C=10	90	89	92	89
CNN-KNN, k = 2	78	82	78	79
CNN-DT	70	70	70	70
CNN-RF Trees = 5	75	76	75	70
CNN-logistic regression Penalty= 'l2', C= 50	85	85	85	85
CNN Only	87	87	87	85

Table 15: The result of extracting features from the second convolutional layer for arousal levels (low, medium, high).

Classifier	F1	Precision	Recall	Accuracy
CNN- RPF-SVM Gamma=0.01, C=10	67	66	70	70
CNN-KNN, k = 2	77	80	72	79
CNN-DT	59	64	67	60
CNN-RF, Trees = 5	50	50	54	60
CNN-logistic regression Penalty= 'l2', C= 50	82	81	84	82

Table 16: The result of extracting features from the second convolutional layer for the binary arousal levels (low, high).

Classifier	F1	Precision	Recall	Accuracy
CNN- RPF-SVM Gamma=0.01, C=10	74	74	74	74
CNN-KNN, k = 2	76	77	76	77
CNN-DT	54	54	55	54
CNN-RF, Trees = 5	65	77	67	69
CNN-logistic regression Penalty= 'l2', C= 50	83	73	74	73

Table 17: The result of extracting features from the FLC for arousal levels (low, medium, high).

Classifier	F1	Precision	Recall	Accuracy
CNN- RPF-SVM Gamma=0.01, C=10	77	85	79	79
CNN-KNN, k = 2	66	65	66	62
CNN-DT	62	64	61	61
CNN-RF, Trees = 5	53	54	53	60
CNN-logistic regression Penalty= 'l2', C= 50	73	74	78	74

Table 18: The result of extracting features from the FLC for the binary arousal levels (low, high).

Classifier	F1	Precision	Recall	Accuracy
CNN- RPF-SVM Gamma=0.01, C=10	69	79	70	72
CNN-KNN, k = 2	64	64	64	64
CNN-DT	54	55	55	55
CNN-RF, Trees = 5	61	61	61	62
CNN-logistic regression Penalty= 'l2', C= 50	81	85	81	81

5.7.4 The Confusion matrices for the highest accuracy

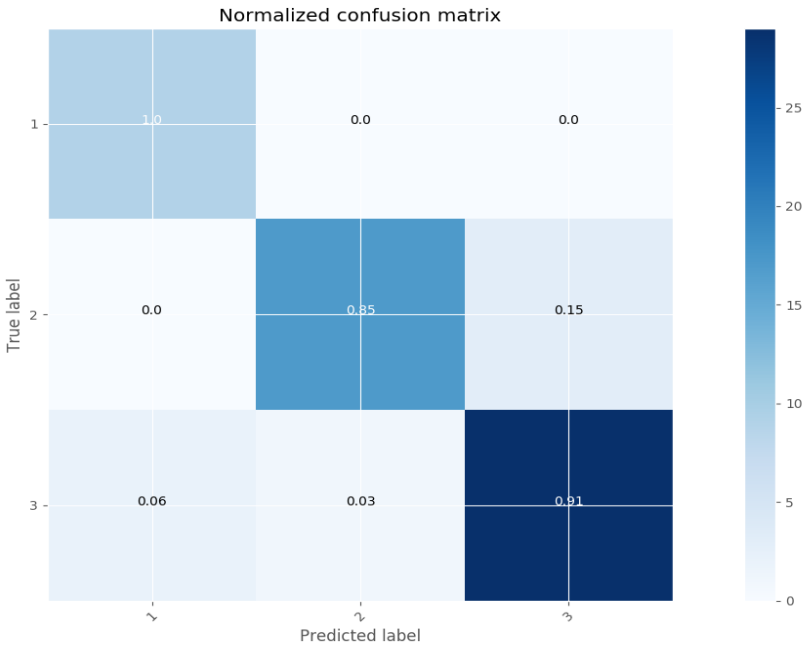

Figure 13. Confusion matrix of the three arousal levels classification using features map of the third convolutional layer with CNN-SVM.

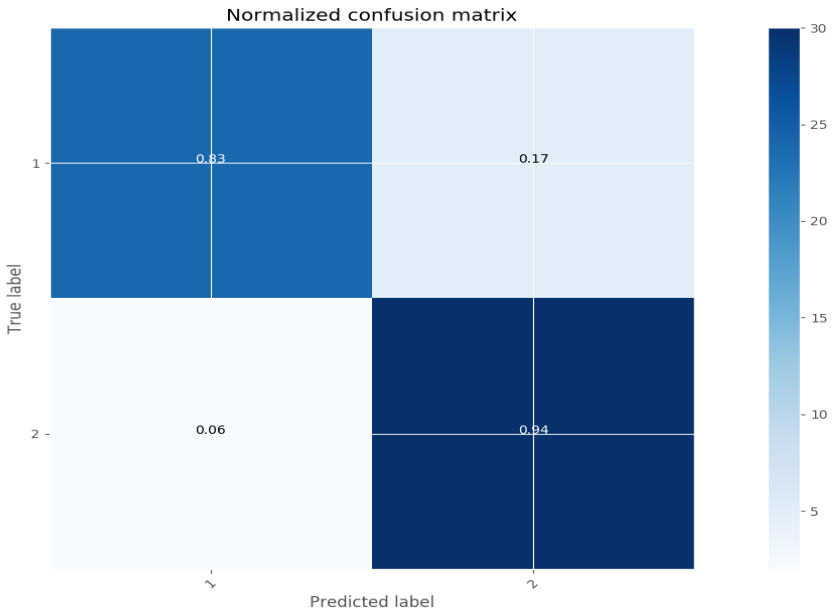

Figure 14. Confusion matrix of the binary arousal levels classification using features map of the third convolutional layer with CNN-SVM.

5.7.5 Comparison with the State-of-the-Art Methods

A study was conducted using the same dataset with traditional machine learning approaches. The dataset was processed by extracting the mean and standard deviation along with temporal, frequency and domain based features. They performed several feature selections methods like the sparse technique. They examine the dataset based on the selected features and they run the machine learning classifier with a combination of the domain based features and select the features with the highest accuracy. The highest accuracy they score is 80 % using binary arousal level [39]. Using features from the third convolutional layer with SVM our model accuracy score is 90 % using three arousal level and 89 % using binary arousal level.

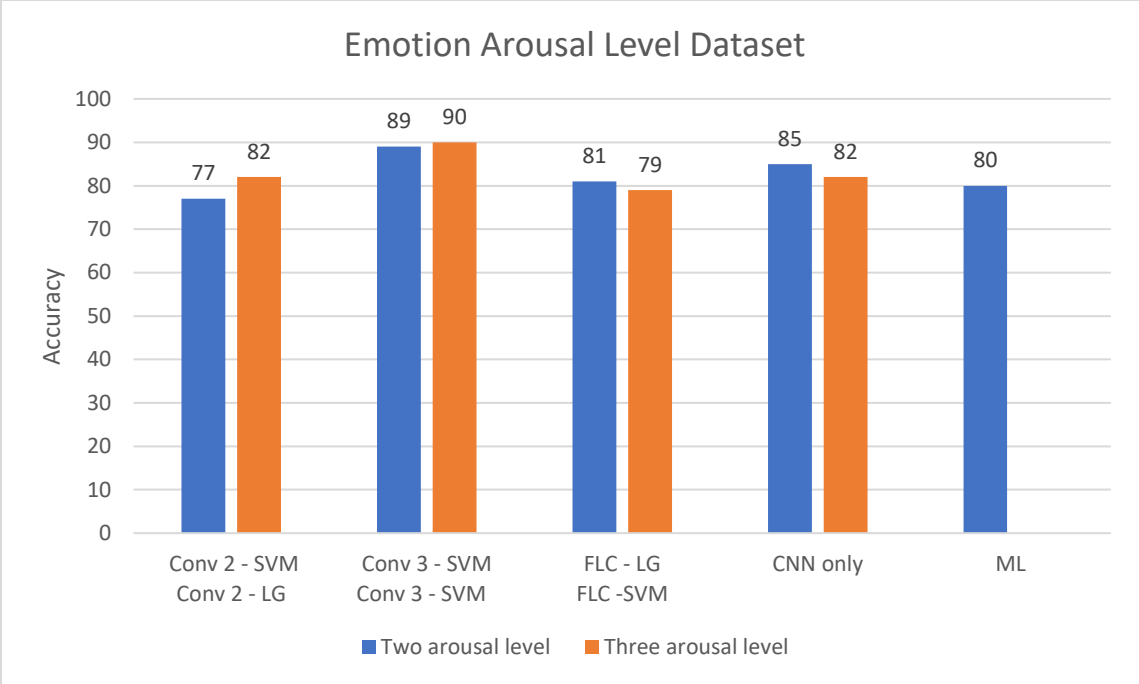

Figure 15. Summary of Experiment Three Results.

5.8 Dataset (DEAP Dataset)

DEAP Dataset contains data that have been collected from 32 subjects. The data were recorded as each subject watched 40 one-minute long excerpts of music videos. Subjects rated each video in terms of the levels of arousal, valence, like/dislike, dominance and familiarity using 1-9 scale. The data consist of 40 channels, 32 of them are EEG channels and the remaining 8 are EOG Horizontal and Vertical, EMG Zygomaticus and Trapezius, GSR, Respiration belt, Plethysmograph and Temperature. Data collected at 512Hz and down-sampled to 128 HZ. The labels of this data are 1-9 scale for each class of the four valences, arousal, dominance and liking [40].

5.8.1 Training Parameters

The model was trained with batch of size of 125. The Adam optimizer's parameters lr, beta1, and beta2 were set to 0. 001, 0.9 and 0.999 respectively. Then the whole model was fine-tuned using the sequential training set. Specifically, we equally split the sequences of 60-s epochs from each subject data into 125 sub-sequences (i.e., batch size). Then we fed 20 epochs from each sub-sequence yielding 2500 epochs per one training step. It should be noted we used the default value for most optimizer parameters such as beta1, beta2. We experimented with the batch size (from 100 to 200) during the training, the epochs (from 10 to 100), the learning rates (from 10^{-1} to 10^{-6}), and l2 weight decay (from 10^{-1} to 10^{-6}). In this dataset, we use batch normalization and dropout layers with penalty 0.5 after last convolutional layers and 0.7 after the fully connected layer.

5.8.2 Experiment four the Model Specification

Table 19: The model specification for the DEAP dataset.

Conv layer 1	Filter: 1*20, Number of channels: 40, Number of filters: 40, Stride (1,1)
Max Pool 1	Kernel size (2, 2), Stride (2,2)
Conv layer 2	Filter: 5* 60, Number of channels:40, Number of filters: 60, Stride (2,2)
Max pool 2	Kernel size (2, 2), Stride (2,2)
Conv layer 3	Filter: 5 * 70, Number of channels: 60, Number of filters: 80, Stride (4,4)
Max pool 3	Kernel size (2, 2), Stride (2,2)
FLC	1400

The approximate training time of CNNs with this dataset is 3 hours 34 minutes.

5.8.3 Experiment four Results

Arousal Level Classification

Table 20: The result of extracting features from the second convolutional layer for arousal levels.

Classifier	F1	Precision	Recall	Accuracy
CNN- RPF-SVM C= 1.0, gamma = 0.0001	44	48	48	49
CNN-LG, C= 1.0	39	41	41	50
CNN-KNN, K = 10	40	41	46	51
CNN-DT, Max depth = 50	38	34	32	50
CNN-RF, Trees = 20	40	50	45	46
CNN Only	51	52	51	53

Table 21: The result of extracting features from the third convolutional layer for arousal levels.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM C=1.0,gamma= 0.0001	57	57	58	63
CNN-LG, C= 1.0	60	59	60	63
CNN-KNN, K = 10	57	58	57	60
CNN-DT Max depth = 50	57	55	60	57
CNN-RF, Trees = 20	54	54	54	62
CNN Only	51	52	51	53

Table 22: The result of extracting features from the FLC layer for arousal levels.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM C=1.0, gamma = 0.0001	48	49	48	50
CNN-LG C= 1.0	40	40	44	55
CNN-KNN K = 10	46	47	50	56
CNN-DT Max depth = 50	39	40	43	55
CNN-RF Trees = 20	40	41	44	55

Liking Level Classification

Table 23: The result of extracting features from the second convolutional layer for liking levels.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM, C=1.0, gamma = 0.0001	46	47	52	50
CNN-LG, C= 1.0	49	50	52	58
CNN-KNN, K = 10	53	55	59	60
CNN-DT Max depth = 50	40	45	54	50
CNN-RF, Trees = 20	47	40	58	57

Table 24: The result of extracting features from the third convolutional layer for liking levels.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM C= 1.0, gamma = 0.0001	63	62	66	63
CNN-LG, C= 1.0	65	62	69	67
CNN-KNN, K = 10	62	61	64	65
CNN-DT Max depth = 50	50	53	49	50
CNN-RF, Trees = 20	55	56	55	61
CNN Only	50	52	50	50

Table 25: The result of extracting features from the FLC layer for liking levels.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM C= 1.0, gamma = 0.0001	50	50	49	50
CNN-LG, C= 1.0	56	51	54	57
CNN-KNN, K = 10	55	55	61	62
CNN-DT Max depth = 50	50	44	53	57
CNN-RF, Trees = 20	41	44	46	48

Valence Level Classification

Table 26: The result of extracting features from the second convolutional layer for valence levels.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM C= 1.0, gamma = 0.0001	41	59	46	48
CNN-LG, C= 1.0	45	48	42	50
CNN-KNN, K = 5	45	44	47	52
CNN-DT Max depth = 50	45	44	45	45
CNN-RF, Trees = 20	45	41	49	50

Table 27: The result of extracting features from the third convolutional layer for valence levels.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM C= 1.0, gamma = 0.0001	66	66	65	66
CNN-LG, C= 1.0	70	70	70	71
CNN-KNN, K = 5	63	64	63	63
CNN-DT Max depth = 50	55	56	55	57
CNN-RF, Trees = 20	58	52	64	61
CNN Only	48	48	47	50

Table 28: The result of extracting features from the FLC layer for valence levels.

Classifier	F1	Precision	Recall	Accuracy
CNN-RPF-SVM C= 1.0, gamma = 0.0001	41	45	47	47
CNN-LG, C= 1.0	47	48	51	57
CNN-KNN, K = 5	60	62	63	66
CNN-DT Max depth = 50	46	46	46	46
CNN-RF, Trees = 20	40	45	44	48

5.8.4 The Confusion Matrices for the Highest Accuracy

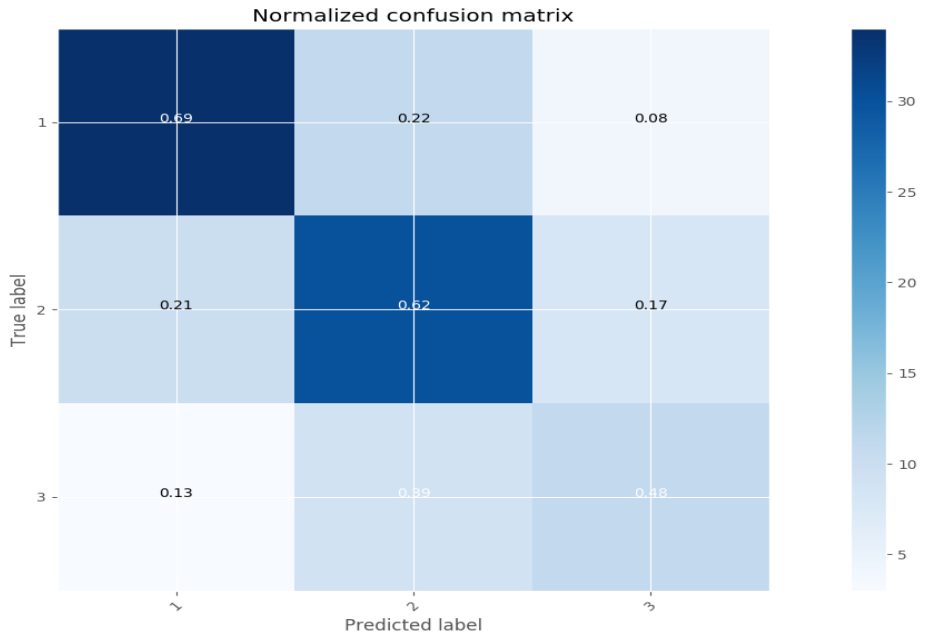

Figure 16. Confusion matrix of the arousal levels classification using features map of the third convolutional layer with CNN-LG.

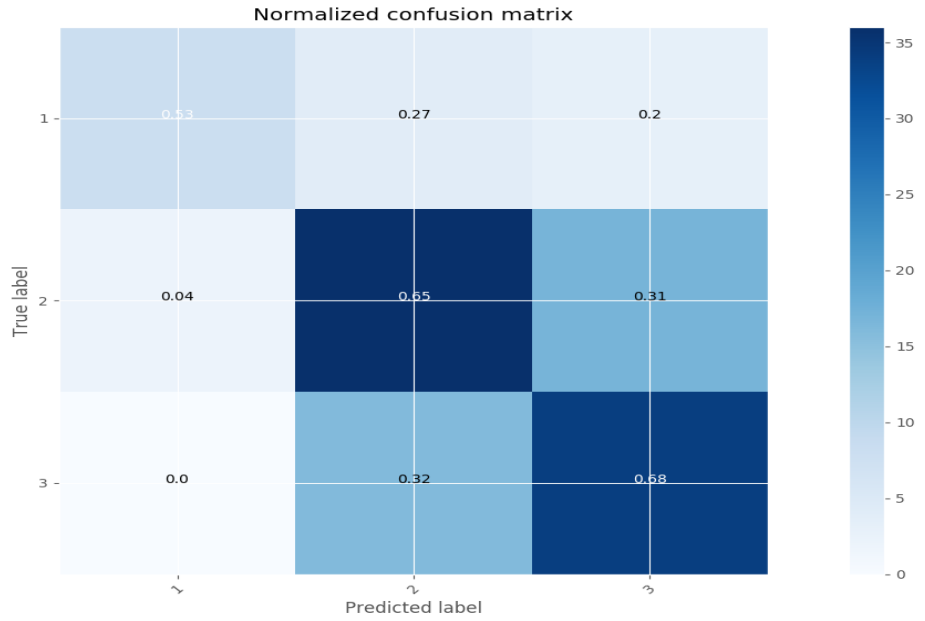

Figure 17. Confusion matrix of the liking levels classification using features map of the third convolutional layer with CNN-LG.

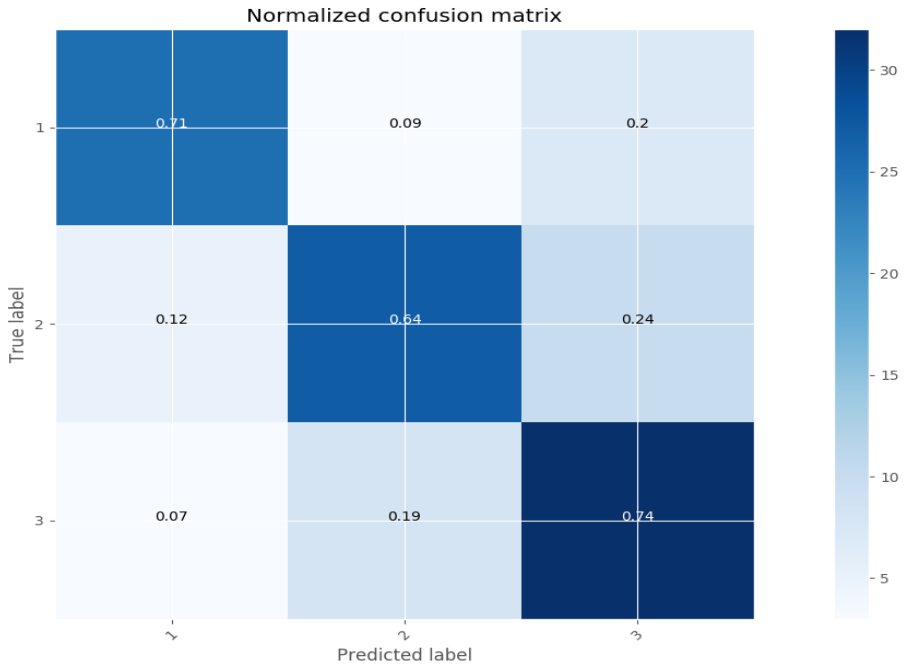

Figure 18. Confusion matrix of the valence levels classification using features map of the third convolutional layer with CNN-LG.

5.8.5 Comparison with the State-of-the-Art Methods

Study was conducted using the same dataset with traditional machine learning approaches. The dataset was processed by down sampling the signals to 256 Hz then a number of features were extracted including the average, standard deviation, spectral power and other features for each signal. Feature selection methods were also applied. They examine the dataset based on the features extracted and they state the accuracy they got by using each set of features. The highest accuracy and F1 they score for the Arousal class is 65 % and 61% respectively. The highest accuracy and F1 they score for the Valence class is 63 % and 60 %. The highest accuracy and F1 they score for the Liking class is 67 % and 63 % [40]. The highest accuracy we score using extracted features from the third convolutional layer. The Accuracy we scored for arousal, valence and liking are 63 %, 71%

and 67 % respectively. Our model performs better in both valence and liking comparing to the state-of-the-art approaches where they utilize hand-engineered method. However, for the arousal level our model accuracy is 2% less accurate.

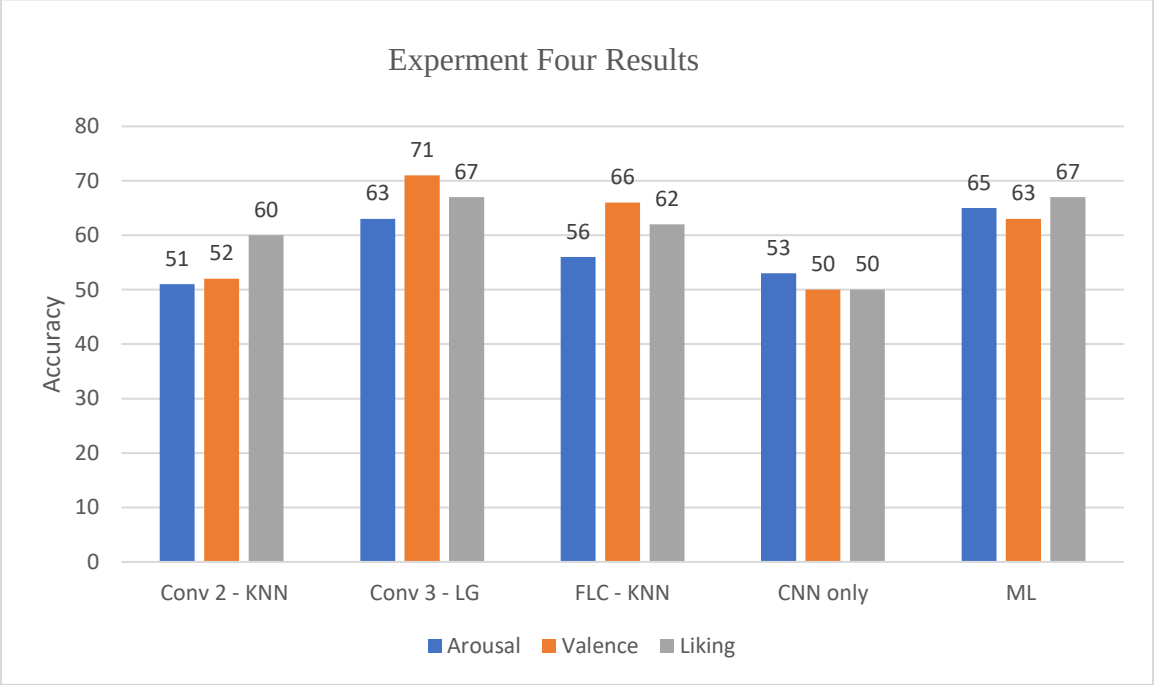

Figure 19. Summary of Experiment Four (DEAP dataset) Results.

5.9 Discussion

The results demonstrated that our model achieved a similar performance on the first and fourth data sets. However, the result also shows that our model outperforms the state-of-the-art methods in second and the third experiments. In the fourth experiment our method performs better in classifying the Valence and Liking emotion levels; however, we score less than the state-of-the-art method for classifying the Arousal levels. We observe that in general, our method outperforms the CNNs and the state-of-the-art performance when we utilize the features which come from the third convolutional layer or the FLC layer. We also observe that the combination of CNNs with SVM or LG score the highest

accuracy in average. The results imply that our model recognized some useful patterns in the signals and generalize well among different subjects. In general, our results showed that our model was able to achieve better performance compared to the state-of-the-art methods.

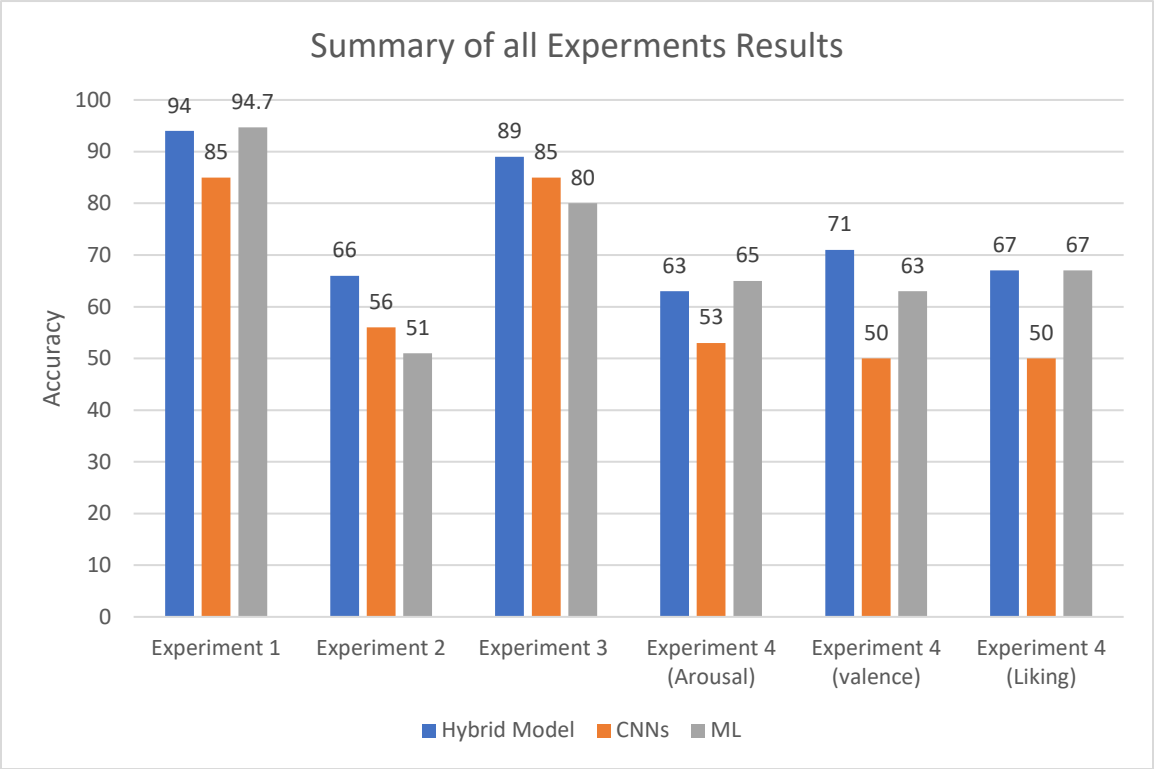

Figure 20. Summary of all the experiments results using our hybrid model, CNNs only and the state-of-the-art machine learning approach.

6. CONCLUSION

In this work we proposed a hybrid model where we combined deep CNNs with traditional machine learning classifiers to eliminate the need for a human expert to study and extract hand-engineered features from biosignals. The main goal of this research is to use deep learning on small datasets without overfitting the training data and we have been able to achieve this goal as demonstrated in our experiments on both small and large datasets. Our model shows promising results on raw biosignals compared to the state-of-the-art approaches that utilize hand-engineered features. Even though our model needs to be fully trained and fine-tuned before applying it to new dataset to extract the best features, we believe that our model is a better approach because our model is able to automatically learn features from the raw datasets. Using a leave-one-subject-out cross validation guarantees that our hybrid model can generalize well on data from different subjects and from different sensors. In this work, the highest performances we got were from either the third convolutional layer or the fully connected layer, however, this may vary depending on the used dataset.

Even though the results are encouraging, and we achieved better performance compared to using deep learning alone or a hand-engineered features approach in the domain of biosignals, it is difficult for humans to understand the features learned by the deep CNNs filters at each layer, which still remains of the disadvantages of neural network-based classifiers.

REFERENCES

[1] Kaniusas, E. Biomedical Signals and Sensors I: Linking Physiological Phenomena and Biosignals. (Heidelberg, Germany: Springer Verlag GmbH, 2012).

[2] Cohen, A. "Biomedical signals: Origin and dynamic characteristics; frequency-domain analysis." The Biomedical Engineering Handbook(Second Edition). CRC Press, 2000.

[3] Janecek, Andreas G. K. , Wilfried N. N. Gansterer, Michael A. Demel, and Gerhard F. Ecker. "On the Relationship Between Feature Selection and Classification Accuracy." University of Vienna (2008).

[4] Tang, Shijian, and Ye Yuan. "Object Detection based on Convolutional Neural Network." Stanford University.

[5] Krizhevsky, Alex, Ilya Sutskever, and Geoffre E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." University of Toronto.

[6] Kim, Yelin, Honglak Lee, and Emily Mower Provost. "Deep learning for robust feature generation in audiovisual emotion recognition." 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (2013).

[7] Roebuck, A., V. Monasterio, E. Gederi, M. Osipov, J. Behar, A. Malhotra, T. Penzel, and G. D. Clifford. "A review of signals used in sleep analysis." *Physiological measurement* 35, no. 1 (2013): R1.

[8] Nakasone, Arturo, Helmut Prendinger, and Mitsuru Ishizuka. "Emotion recognition from electromyography and skin conductance." In *Proc. of the 5th International Workshop on Biosignal Interpretation*, pp. 219-222. 2005.

[9] Kazi, Sheeba S., and Bhavana P. Harne. "Statistical Signal Processing of EEG Signals for Lie Detection." International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, April 4, 2015.

[10] Soman, Karthik, Varghese Alex, and Chaithanya Srinivas. "Analysis of physiological signals in response to stress using ECG and respiratory signals of automobile drivers." In *Automation, Computing, Communication, Control and Compressed Sensing (iMac4s), 2013 International Multi-Conference on*, pp. 574-579. IEEE, 2013.

[11] Mporas, Iosif, Vasiliki Tsirka, Evangelia I. Zacharaki, Michalis Koutroumanidis, Mark Richardson, and Vasileios Megalooikonomou. "Seizure detection using EEG and ECG signals for computer-based monitoring, analysis and management of epileptic patients." *Expert Systems with Applications* 42, no. 6 (2015): 3227-3233.

[12] Brumberg, Jonathan S., Alfonso Nieto-Castanon, Philip R. Kennedy, and Frank H. Guenther. "Brain–computer interfaces for speech communication." *Speech communication* 52, no. 4 (2010): 367-379.

[13] Naït-Ali, Amine, ed. *Advanced biosignal processing*. Springer Science & Business Media, 2009. [single processing]

[14] Tsinalis, Orestis, Aul M. Matthews, Ike Guo, and Stefanos Zafeiriou. "Automatic Sleep Stage Scoring with Single-Channel EEG Using Convolutional Neural Networks." (2016).

[15] Martinez, Hector P., Yoshua Bengio, and Georgios N. Yannakakis. "Learning deep physiological models of affect." IEEE Computational Intelligence Magazine 8.2 (2013): 20-33.

[16] Stober, Sebastian, Avital Sternin, Adrian M. Owen, and Jessica A. Grahn. "3) Deep Feature Learning For EEG Recordings." The Brain and Mind Institute, University of Western Ontario (2016).

[17] Walker, Ian. "Deep Convolutional Neural Networks for Brain Computer Interface using Motor Imagery." Imperial College of Science Technology and Medicine (2015).

[18] Hubel, David H., and Torsten N. Wiesel. "Receptive fields and functional architecture of monkey striate cortex." *The Journal of physiology* 195, no. 1 (1968): 215-243.

[19] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

[20] Gawehn, Erik, Jan A. Hiss, and Gisbert Schneider. "Deep learning in drug discovery." *Molecular informatics* 35, no. 1 (2016): 3-14.

[21] Bergstra, James, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, Ian Goodfellow, Arnaud Bergeron, Yoshua Bengio, and Pack Kaelbling. "Theano: Deep learning on gpus with python." (2011).

[22] Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin et al. "TensorFlow: A System for Large-Scale Machine Learning." In *OSDI*, vol. 16, pp. 265-283. 2016.

[23] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Convolution Networks " In Deep learning. MIT press, 2016.

[24] Schmidhuber, Jürgen. "Deep learning in neural networks: An overview." *Neural networks* 61 (2015): 85-117.

[25] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In *International Conference on Machine Learning*, pp. 448-456. 2015.

[26] Chen, Yushi, Hanlu Jiang, Chunyang Li, Xiuping Jia, and Pedram Ghamisi. "Deep feature extraction and classification of hyperspectral images based on convolutional neural networks." *IEEE Transactions on Geoscience and Remote Sensing* 54, no. 10 (2016): 6232-6251.

[27] Ng, Andrew Y. "Feature selection, L 1 vs. L 2 regularization, and rotational invariance." In *Proceedings of the twenty-first international conference on Machine learning*, p. 78. ACM, 2004.

[28] Dahl, George E., Tara N. Sainath, and Geoffrey E. Hinton. "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. IEEE, 2013.

[29] Srivastava, Nitish, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research* 15, no. 1 (2014): 1929-1958.

[30] Krizhevsky, Alex. "One weird trick for parallelizing convolutional neural networks." *arXiv preprint arXiv:1404.5997* (2014).

[31] Dixon, Matthew Francis, Diego Klabjan, and Jin Hoon Bang. "Classification-based Financial Markets Prediction using Deep Neural Networks." (2016).

[32] Ngiam, Jiquan, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V. Le, and Andrew Y. Ng. "On optimization methods for deep learning." In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 265-272. 2011.

[33] Tieleman, Tijmen, and Geoffrey Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." *COURSERA: Neural networks for machine learning* 4, no. 2 (2012): 26-31.

[34] Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).

[35] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. "Optimization for Training Deep Models " In Deep learning. MIT press, 2016.

[36] Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin et al. "TensorFlow: A System for Large-Scale Machine Learning." In *OSDI*, vol. 16, pp. 265-283. 2016.

[37] Rampasek, Ladislav, and Anna Goldenberg. "Tensorflow: Biology's gateway to deep learning?." *Cell systems* 2, no. 1 (2016): 12-14.

[38] Micucci, Daniela, Marco Mobilio, and Paolo Napoletano. "UniMiB SHAR: a new dataset for human activity recognition using acceleration data from smartphones." *arXiv preprint arXiv:1611.07688* (2016).

[39] Hinkle, lee. "Determination of emotional state through physiological measurements". (2016)

[40] Koelstra, Sander, Christian Muhl, Mohammad Soleymani, Jong-Seok Lee, Ashkan Yazdani, Touradj Ebrahimi, Thierry Pun, Anton Nijholt, and Ioannis Patras. "Deap: A database for emotion analysis; using physiological signals." *IEEE Transactions on Affective Computing* 3, no. 1 (2012): 18-31.