Convolutional Neural Networks and Extreme Learning Machines on Mobile Devices

Baris Sozudogru

Bachelor's thesis

Convolutional Neural Networks and Extreme Learning Machines on Mobile Devices

Baris Sozudogru

May 3, 2021

Chair of Data Processing
Technische Universität München

Baris Sozudogru. *Convolutional Neural Networks and Extreme Learning Machines on Mobile Devices.* Bachelor's thesis, Technische Universität München, Munich, Germany, 2021.

Supervised by Prof. Dr.-Ing. Klaus Diepold and M.Sc. Matthias Kissel; submitted on May 3, 2021 to the Department of Electrical and Computer Engineering of the Technische Universität München.

Chair of Data Processing, Technische Universität München, 80290 München, Germany, http://www.ldv.ei.tum.de/.

Abstract

In the last decade, there were several important advancements in the field of deep learning. The impact of the Convolutional Neural Networks (CNN) on the advancements in the field is huge, and CNN architectures serve as a crucial solution for many machine learning problems. The accuracy of the method, particularly on image classification tasks, has no equivalent. However, the method suffers from high training times. On the other hand, Extreme Learning Machine is an algorithm, which requires significantly less training times and can achieve comparable accuracy.

This thesis tests and compares the performances of three methods, Convolutional Neural Networks, Extreme Learning Machines, and a combination of the two, through experiments on image classification tasks using popular image data sets. Two types of devices are used: computer and to represent mobile platforms Raspberry Pi.

The experimental results in this thesis confirm the benefits of using Extreme Learning Machines, particularly on comparatively complex datasets. Overall, mobile platforms provide comparable accuracy for image classification tasks and acceptable training times.

The implementation of the Extreme Learning Machines algorithm seems beneficial, particularly on less computationally powerful devices such as mobile platforms. Its utility warrants to be tested on a range of devices, tasks, and data sets, in future studies.

Keywords: Convolutional Neural Network (CNN), Extreme Learning Machine (ELM), Raspberry Pi, image classification

Table of Contents

List of Abbreviations

AdaGrad	Adaptive Gradient
Adam	Adaptive Moment Estimation
API	Application Programming Interface
ARM	Acorn RISC Machine
BiT	Big Transfer
CISC	Complex Instruction Set Architectures
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
ELM	Extreme Learning Machine
GPIO	General-Purpose Input/Output
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IDLE	Integrated Development and Learning Environment
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoT	The Internet of Things
ISA	Instruction Set Architecture
MLP	Multilayer Perceptron
OS	Operating System
PCB	Printed Circuit Board
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RMSprop	Root Mean Square Propagation
RVFL	Random Vector Functional Link
SBC	Single Board Computer
SGD	Stochastic Gradient Descent
SLFN	Single Layer Feed-Forward Neural Network
SoC	System on a Chip
SSH	Secure Shell Protocol
TELM	Two-Hidden-Layer Extreme Learning Machine
TLFN	Two-Hidden Layer Neural Network

List of Figures

1. Introduction

Deep learning has achieved extraordinary progress in recent times. As a standing out deep learning architecture, Convolutional Neural Network (CNN) has been instrumental in the process. CNN's are used in several areas of expertise, such as natural language processing, image classification, speech recognition, and image reconstruction. It has already proven its success in various fields.

The research in the field is still mostly focused on improving accuracy, although computational systems have already achieved a high accuracy on classification tasks. Currently, the results for artificial intelligence systems are way beyond human performance. The success is apparently evident from the decrease of the error rates for the ImageNet dataset [1]. In 2012, when the AlexNet [2] was firstly introduced, it had a 16.4% top-5 error for the task, where human performance is 5% on the task. In 2015 ResNet-152 surpassed human-level performance with 3.57% top-5 error. SENET [3], the winner of the ILSVRC-2017 (ImageNet Large Scale Visual Recognition Challenge), achieved 2.251% top-5 error. Moreover, shifting the aspect of research from accuracy optimization to performance optimization may be beneficial for the energy consumption on large-scale operations. Likewise, research on the performance of computationally less powerful platforms would contribute to the subject as well.

Even though the CNN architectures have significantly high accuracy, which is sufficient for most of the tasks, the training times for the models can be exceptionally high. The high value of training times is primarily due to backpropagation. The backpropagation algorithm is an iterative algorithm, and the iteration process can become extremely time-consuming in case of complex tasks. Since CNN architectures have lots of parameters, e.g., AlexNet [2] has 62,378,344 parameters, the computational complexity for the task is extremely high. Furthermore, the training times for the method increase in parallel to the number of layers. Moreover, the generalization ability of CNNs is limited due to fully connected layers for the classification [4].

ELMs [5], on the other hand, perform well regarding the training times. The algorithm has high generalization performance and performs well with a high amount of data. Besides, the algorithm maintains a good level of accuracy even with small datasets. In addition, the combination of CNN and ELM may be another option.

The performance improvements are critical, particularly for implementation on computationally less powerful devices. Current mobile platforms can be included in this group. Mobile platforms have become an integral part of daily life. Consequently, deployment of the CNN models on mobile platforms is gaining popularity. On the other hand, training the models on mobile platforms seems challenging at first sight. However, there are several motivations for the training. With the increasing privacy concerns of the community, local model training might gain importance. In the current state, people are concerned about the utilization of their data from big tech companies. For example, the percentage of people

revealing information on social platforms decreases over time [6]. Moreover, connection unavailability is an important constraint for mobile platforms; in those conditions, the local training of devices is practical. Particularly the developments in the area of transfer learning [7] are exciting, considering the impressive performance of Big Transfer (BiT) [8], and the implementations for the mobile platforms seem promising, which has the potential to become a motivation for local training of models. Besides, from the economic perspective, less data transfer cost can be achieved with local training.

For examining the performance of deep learning tasks on mobile platforms, Raspberry Pi can be considered a bridge. From the software development aspect, it is compatible with deep learning frameworks and provides independence with its unique operating system: Raspberry Pi OS (Operating System), which is Debian-based. From the hardware perspective, it has many similarities with the mobile platforms, including CPU (Central Processing Unit) architectures, chip designs, etc. Furthermore, Raspberry Pi has important areas of application in Internet of Things (IoT) [9], and the involvement of deep neural networks in IoT is increasing.

One of the most important aspects of training neural network models on mobile platforms is the optimisation of the codes. Therefore, for the implementation of models, highly optimized frameworks and libraries are preferred.

In this thesis, training time and accuracy of CNN, ELMs, and a combination of CNN and ELM were examined. In particular, methods with the potential to decrease training times and maintain the test accuracy at the same time were examined. The performances of the models were tested on a mobile platform and a computer. The objective of the experiments was to test the methods to improve the performance of image classification on a computationally weak system.

2. Background

2.1. Related Work

To the best of my knowledge, there is no research comparing the performances of CNNs, ELMs, and the combination of CNN and ELM on mobile platforms, particularly on Raspberry Pi. However, there is research previously done, for combining CNNs and ELMs and comparing the training performance of models on computers. In previous hybrid models, the approaches of combining the components differ.

In the paper by Yoo et al. [10], the parallel ELM learning was applied to the convolutional layers of CNNs, layer by layer. Each output feature map of each convolutional layer was reshaped so that ELM learning can be applied to convolutional layers. In the process, convolutional filters were determined analogously to the analytical calculation of the output weight matrix for feed-forward neural networks. The paper reports that the CNN-ELM method improves the missing rate of the implemented task, achieves 20 times faster training compared to CNN with the backpropagation algorithm, and CNN performs better than ELM. However, the performance was compared solely on accuracy.

In a previous work [11], CNN was trained using ELM, based on the idea of auto-encoding. In that method, convolutional filter training is done by the ELM-based auto-encoding approach. The proposed method is a layer-wise training procedure and has significant advantages over conventional CNNs with backpropagation.

In the study by Kim et al. [12], ELM and CNN were combined by adding a layer between convolutional and pooling layers of CNN to utilize ELM. Furthermore, the method applies the ELM algorithm to the last layer of the fully connected neural network in CNN.

Again in another paper [13], the ELM-based auto-encoding approach, inspired by the paper [10], was used for the learning process of convolutional filters. In that paper, the model was modified to get better results. As reported by the paper, the proposed model was up to 950 times faster than the training of CNNs with backpropagation.

There are other studies using CNN as a feature extractor and ELM as a classifier for age prediction using facial images [14], for traffic sign recognition [4], and for wireless capsule endoscopy image classification [15]. The studies support that CNN-ELM structure has better performance. Furthermore, there are also research papers [16], solely comparing the performance of ELM and CNN on image-based classification, with similar conclusions, pointing out the benefits of ELM.

According to the results obtained from the above research, the success of the approach seems promising.

2.2. Machine Learning

Machine learning is focused on building algorithms to discover underlying relations in data and make predictions using statistics. With the algorithms, models are formed based on training data, which consists of samples. The quality and the quantity of the samples play important role for the performance of the models. Furthermore, the learning process highly depends on determining the closeness of the predictions to actual outcomes since adjustments are to be made according to prediction deviations from the actual values.

Machine learning includes different methods for implementation. The methods differentiate regarding the type and complexity of data and task. In the following section, one of these methods will be discussed: Deep Learning.

2.3. Deep Learning

The initial ideas for neural networks started to emerge with the first neural network mathematical model proposed by Walter Pitts and Warren McCulloch in 1943 [17]. Moreover, the first algorithms concerning neural networks originate from the research by Alexey Ivakhnenko and VG Lapa in 1968 [18]. The advancements in the field of neural networks accelerated with the application of backpropagation to neural networks in 1986 [19].

Deep learning is a form of feature learning method, which typically consists of application of neural networks. The term “deep” refers to the presence of multiple layers in the network. The term “Deep learning” is mentioned first in an article in 1986 by Rina Dechter [20], and the start of the current deep learning era is in 2006 [21]. Since then, it is considered one of the most promising research areas of machine learning. It drastically improved the performance of complex tasks such as object recognition and speech recognition [22].

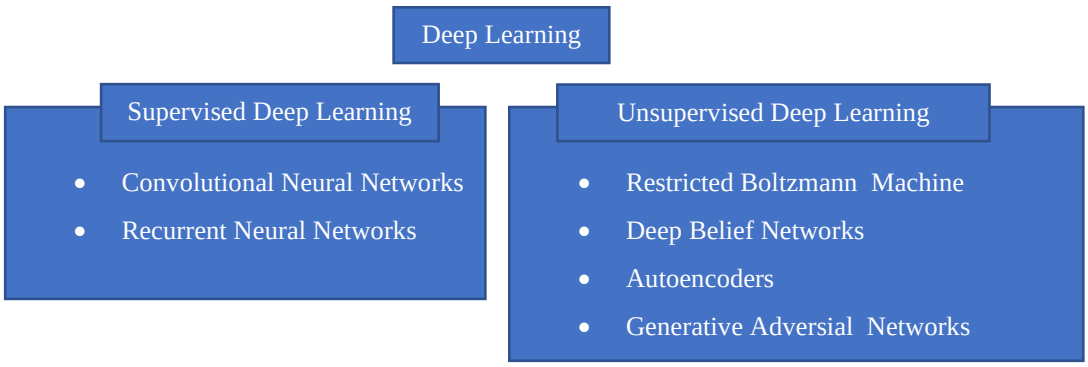

Figure 2.1 Deep learning diagram showing its two forms, supervised and unsupervised deep learning, with their unique structures. Unsupervised learning algorithms discover patterns from data without labels. Supervised learning on the other hand, adapts the system for a given labeled input data to predict the outcome of a later input.

There are two forms of deep learning, supervised and unsupervised deep learning. These two forms of deep learning have their unique architectures. In Figure 2.1, some of the example architectures are shown.

Unsupervised learning algorithms discover patterns from data, which has no labels assigned to it. This means that no supervision for the models is provided. Moreover, unsupervised deep learning algorithms are advantageous since the presence of unlabeled data is frequent in real-world data. The popular examples of unsupervised deep learning architecture include Restricted Boltzmann Machine, Deep Belief Networks, Autoencoders and Generative Adversarial Networks.

Supervised learning aims to adapt the system for a given input data to predict the outcome of given input later. The learning data consists of features and labels. The features are the representation of input data, and labels are the indicator of outcomes [23]. In the following section, an example of it, Convolutional Neural Networks, will be discussed in further detail.

2.4. Convolutional Neural Networks

Convolutional Neural Networks are mostly supervised deep learning architectures. They are considered the most fitting deep learning algorithm for image processing tasks with a large amount of data. Its superior performance comes from the methods used for feature extraction. Instead of having each pixel of the image as a feature, it captures relations in the image by using filters and deal with the important features. The process results in a significant decrease in the number of parameters. The inspiration for these methods of CNN's comes from the visual cortex of humans. In the brain, there are different neurons, stimulating different features of the precepted image. Some regions of neurons are stimulated only in the presence of certain structural features. For instance, the horizontal edge of an image stimulates another region of neurons than the vertical edges of an image [23].

The research in the field goes back to the works of D.H. Hubel and T.N. Wiesel in 1962 and plays a crucial role in the development of CNNs [24]. The developments in the field of neural networks are highly influenced by the advancements in cognitive science, and the connection in between is getting stronger. With the developments in technology, CNNs are used today, e.g., for complex brain image analysis, and contribute to developments in cognitive sciences [25].

The first inspiration for CNN's, Neocognitron, is proposed by Kunihiko Fukushima in 1980 [26]. The neural network designed by him had a multilayered design, pooling layer, and convolutional layer, and was suitable for visual tasks. Later, in the 1990s Yann LeCun et al. review methods applied to handwritten character recognition and compare the methods. They showed that CNN can be used for the tasks and reports that CNN's outperform any other method used on the tasks [27]. LeCun et al. used MNIST database [28] for the tasks, and today it is the most popular dataset for image processing systems. The models covered in this thesis also include training with MNIST dataset.

In today's information age, the availability and accessibility of datasets increased greatly. Furthermore, more advanced hardware devices are developed to support much more complex CNN architectures, and the developments in those fields are followed by the developments in CNN architectures. Today's recent and prominent architectures include: AlexNet [2] in 2012, VGG [29] in 2014, GoogleNet (InceptionNet) in 2014 [30], ResNet [31] in 2015, DenseNet [32] in 2017, and Big Transfer (BiT) [8] in 2019. For instance, BiT achieves 87.5% accuracy on ILSVRC-2012 (ImageNet Large Scale Visual Recognition Challenge), 99.4% on CIFAR-10 dataset. In the past, e.g., in comparison to LeNet [27], human performance was better than CNN's, but today performance of CNN's is far beyond the performance of humans. Some of the mentioned architectures are developed by research groups, which are part of big industrial companies. The motivations for scientific research and industrial investment in the field are immense.

CNN has significant advantages over its predecessors. However, there are some challenges for the training of CNNs. With the development of modern neural network architectures comprising of very big architectures, which can solve complicated tasks, the number of parameters required is increased. Even though CNN is a highly efficient method compared to standard neural networks, training times can reach high levels even for CNNs. The main reason for that is the process of updating the weights, which is done through backpropagation. In this thesis, with the combination of CNN and ELM, the problem with the high training time is tried to be solved, and its benefits are tried to be shown.

Neural networks at their core consist of neurons; therefore, a brief introduction is necessary. A single neuron might have multiple inputs (x_i) and an output (y). Each input has a weight (w_i) associated to it, and for each neuron, a bias (Θ) is added. Later, an activation function is applied. In neural networks, the output of a single neuron can be determined with:

$$\sigma\left(\sum_i w_i x_i + \Theta\right) = y \tag{1}$$

where σ represents the activation function and i is the number of inputs. Output (y_j) of the j^{th} neuron can be calculated using the formula:

$$\sigma\left(\sum_i w_{j,i} x_i + \Theta_j\right) = y_j \tag{2}$$

Furthermore, CNN's can be partitioned and examined in 6 different layers:

- Input Layer
- Convolutional Layer(s)
- Pooling Layer(s)
- Flatten layer

- Fully connected layer(s)
- Output layer

Each layer is represented in Figure 2.2 below.

CNN's consist of two main stages: feature extraction and classification stages. Input layer, convolutional layer(s), and pooling layer(s) are considered in feature extraction stage. On the other hand, flatten layer, fully connected layer(s), and output layer are used for classification.

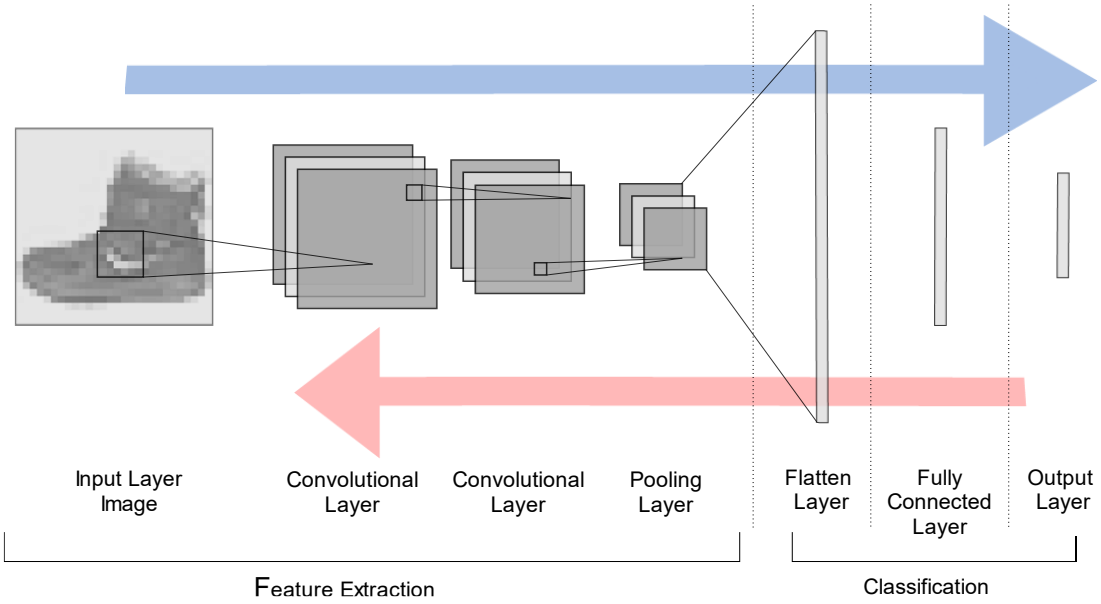

Figure 2.2 Convolutional Neural Network Structure. The arrows represent the propagation direction. Blue arrows indicate forward propagation and red arrow represents backpropagation direction.

The operations in the CNN are conducted from input layer to output layer, typically in the same order. In different architectures, pooling layer can also be placed between convolutional layers. However, in the thesis experiments, the above architecture is preferred. Mostly, between each layer, an activation function is applied. Activation functions provide non-linearity, and each type of them has its own properties. Flatten layer is the flattened output of the pooling layer.

2.4.1. Layers and Components

Convolutional neural networks are a crucial part of the research and should be examined profoundly. A layer-wise examination of the CNN is beneficial. We are starting with the most crucial and distinguished layer: convolutional layer.

2.4.1.1. Convolutional Layer

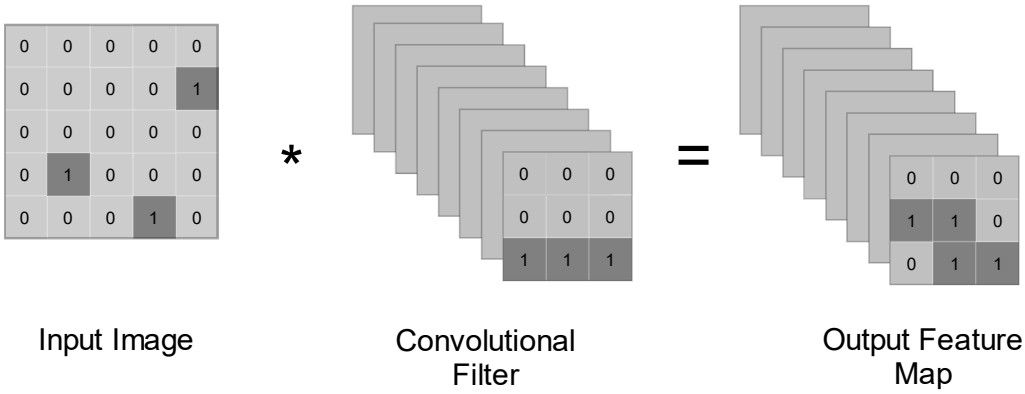

Figure 2.3 Convolution Operation

Convolutional Layers are the most distinctive part of CNN. The main task of the convolutional layer is to detect features within local regions of the input image. Detected features generally represent the structures, which are common in the input image [23]. In Figure 2.3 convolution operation is shown. The input layer, in this case, has only one channel, is convolved with filters, in this case, eight filters. A bias is added, and then an activation function is applied. Subsequently, the results are transferred into feature maps. Each filter corresponds to a feature of the input image and creates a feature map.

Moreover, size of the feature map is calculated with the formula:

$$O = \frac{I - F + 2P}{S} + 1 \tag{3}$$

Here in the formula, "O" represents the size of the output feature map, "I" represent the size of input image, "F" represents the size of the filters, "P" represents padding, and "S" represents stride.

For example, if the input image has the dimensions of 28 x 28 and a convolutional layer follows with a filter size of 5 (5x5), a stride of 1, and without padding, then the size of the feature map will be $\frac{28-5+0}{1} + 1 = 24$.

Hyperparameters concerning the convolutional layer:

- **Filter size:** Filter size represents the dimensions of filters. Filter size of 5 means a size of 5x5 in the convolution operation. In the experiments filter size of 5 is preferred.
- **Number of filters:** Number of filters represents the different structures that can be extracted from the input image. In the experiments, eight filters are used for each convolutional layer.
- **Padding:** It is used to reduce over shrinkage of the image caused by the repeated use of a convolutional layer. In the case of "valid padding", the padding value is set to

zero. The “same padding” is used to make the dimensions of the feature map the same as the input feature map.

- **Stride:** Describes how much the filters are shifted through input feature map when convoluting (vertically and horizontally). The size of the output feature map becomes smaller as the stride increases. Moreover, as the stride size becomes too small, the overlapping of the receptive fields increases, and the output feature map expands.

In the matter of convolution on 3d images, which consists of 3 channels, there should also be three channels for the filters. The result of the convolution of input (consists of 3 channels) with filter (consists of 3 channels) is output feature map with a single channel. In most cases the three channels consist of red, blue, and green (RGB) values of the image. A visual representation of it can be found in Figure 2.4. MNIST [28] and Fashion MNIST [33] datasets consist of one channel, and CIFAR-10 [34] datasets consist of 3 channels. All three datasets are widely used for deep learning tasks. The MNIST and Fashion MNIST datasets are used in the experiments of the present thesis for the image classification tasks.

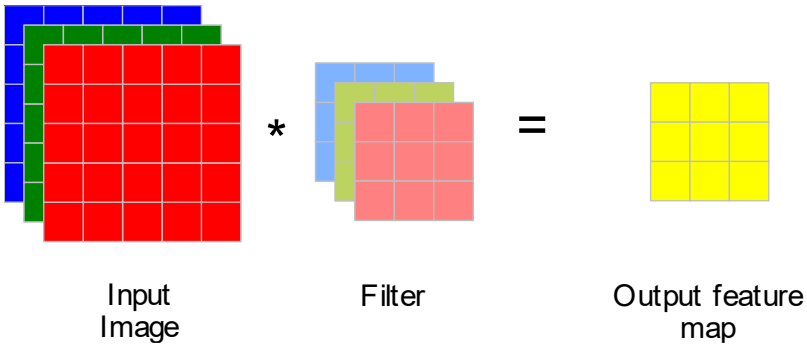

Figure 2.4 3-channel Convolution Operation. A 3x3 filter with three channels is convoluted over an input image consists of three channels, and the output feature map is obtained.

2.4.1.2. Pooling Layer

Due to the high number of parameters in CNN, computational load is immense. Therefore, to reduce computation, a pooling layer is applied, which reduces the size of feature map. It is also called down-sampling layer. In most cases, pooling layer follows convolutional layer and application of activation function to it. Frequently, pooling layer is applied before the feature map is flattened and enters fully connected layer for the classification. It is also used between convolutional layers.

There are different types of pooling operations:

- Minimum Pooling
- Maximum Pooling
- Adaptive Pooling
- Average Pooling

Maximum pooling is the most common operation. The reason is that maximum pooling performs well when extracting extreme features, which is beneficial ,e.g., edges. The operation is visualized in Figure 2.5 below. A max-pooling layer, typically a size of 2x2, is applied to a region on the input feature map. The input with the highest value in the region is inserted in the output feature map. The max-pooling layer is shifted through the input feature map. The shifting amount is determined by the stride, which is typically two. The dimensions of the output feature map can be calculated with the same formula used for convolutional layers. For example, if the input feature map has the dimensions of 4x4 and the max-pooling kernel (also mask size) is 2x2, then $\frac{4-2}{2} + 1 = 2$ is the size of the output feature map. When applying pooling operation on input feature map with three channels, number of channels is preserved, as shown in Figure 2.6. The padding might be necessary in the case of odd size of input feature map. Otherwise, the information on last row and column will be lost.

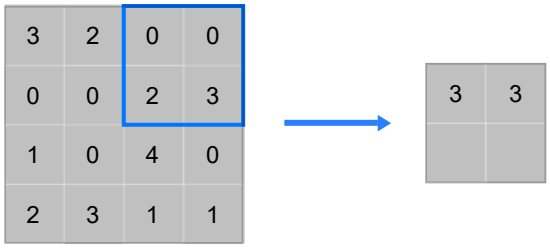

Figure 2.5 Max-Pooling Operation

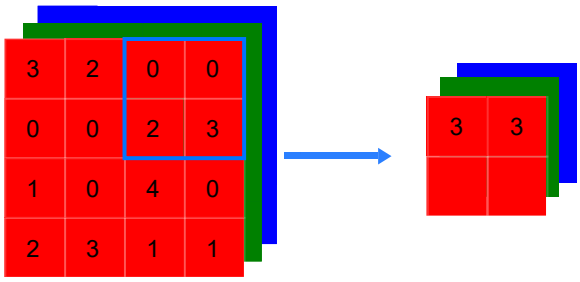

Figure 2.6 Max-Pooling Operation with 3 Channels

It is to be noted that the weights of the convolutional layers are updated during the backpropagation, but pooling layers remain the same. There is no extra computational load through backpropagation with pooling layers.

2.4.1.3. Fully Connected Layers

The first two layers introduced in the previous sections are used for feature extraction. In this section, fully connected layers will be discussed, which are responsible for classification. In this stage, the features are combined into attributes to make classification of the images. A visual representation of fully connected layers is shown in Figure 2.7.

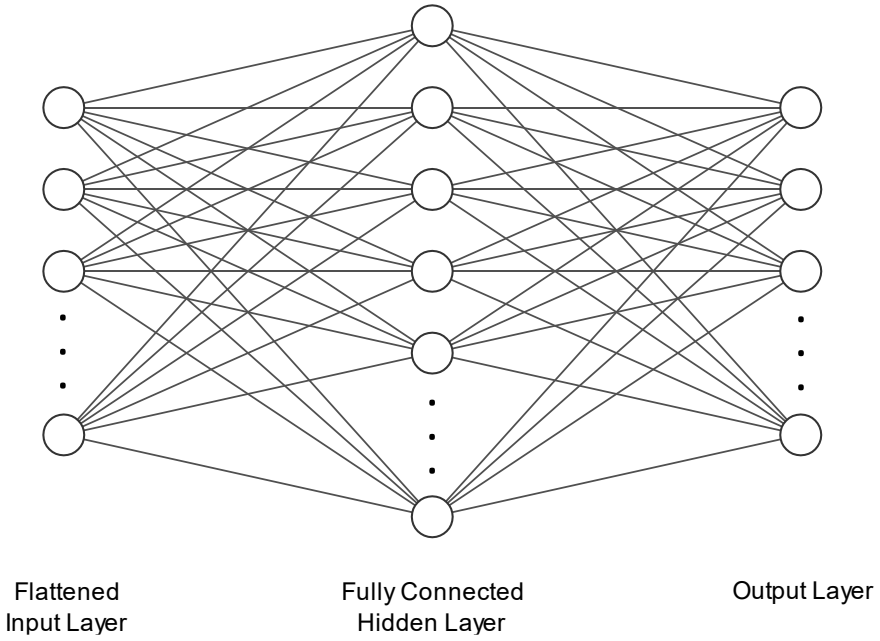

Figure 2.7 Fully Connected Layers

As the name implies, all the nodes in the hidden layer must be connected with all the nodes in the fully connected output layer. The number of hidden layers may vary. The fully connected layers in CNNs can be considered as a multilayer perceptron (MLP). The critical difference from standard multilayer perceptron is that the output of the previous layers should be flattened before taken as input.

The Flattened Input Layer consists of a flattened output feature map of the pooling layer (considering the structure in Figure 2.2). The dimension of the flattened input layer can be calculated through the output feature map of pooling layer with the formula:

$$N \times W \times H \tag{4}$$

"N" refers to the number of output feature maps of the previous layer (convolutional layer or pooling layer), "W" refers to the width of output feature map, and "H" refers to the height of the output feature map. For example, if the output feature map has the dimensions of 24x24 size and number of 8 layers, then the dimension of fully connected input layer is 4608.

After the fully connected output layer, Softmax operator is used, which is suitable for multi-classification tasks. After the application of Softmax, the class probability distributions are normalized in a range of 0 to 1. It is to be noted that also other activation functions can be used for the output layer.

A more detailed discussion on Softmax and the corresponding formula (equation 6) can be found in the next section.

2.4.2. Activation function

In this section, activation functions are introduced. Activation functions in neural networks are functions outputting the result of applying mathematical equations on its input, considering a threshold behaviour. It is originally inspired by the action potential of biological neurons. They are crucial component of neural networks and provides non-linearity in CNNs. Non-linearity is essential for intricate tasks, which have mostly non-linear dynamics, and without activation function, it is not provided. Other operations used in CNNs are linear. When using gradient-based methods, the activation functions in general should be differentiable and are desired to be continuous on every point [23].

There are several activation functions used with CNNs:

- ReLu Activation Function
- Softmax Activation Function
- Tanh Activation Function
- Sigmoid Activation Function

The visual representations of the activations function can be seen below in Figure 2.8. Softmax is a multivariable function. Therefore, it is not feasible to display it on two-dimensional plots.

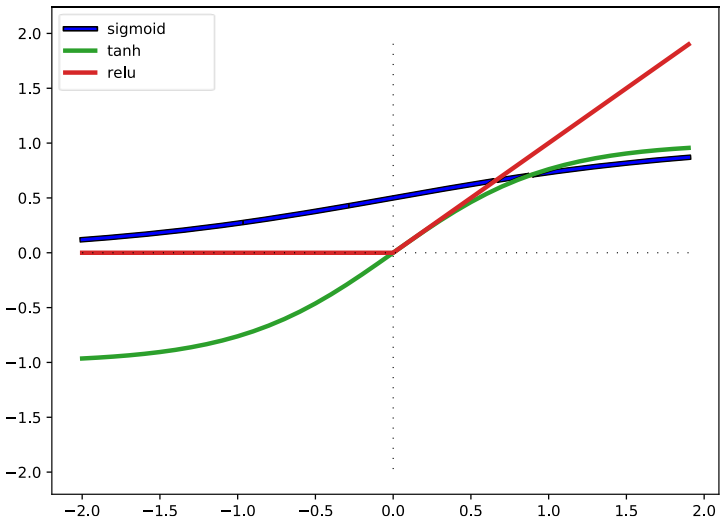

Figure 2.8 Activation Functions. The graph shows the representation of three activation functions in 2-dimensional space. The blue line represents sigmoid activation function, the red line shows ReLu activation function, and the green line describes tanh activation function.

The selection of the activation function plays a crucial role in the dynamics of CNNs. In CNN's, ReLu activation function is applied primarily after the convolutional operation in the convolutional layer. In that case, the input of the activation function is the output feature map of convolutional layer, and the output of the activation function is the output activation map, which ensures the non-linearity in the model. ReLu can be used in fully connected layer as well. Neural network training times with ReLu are better in comparison to Sigmoid and Tanh

activation functions [23]. However, recently introduced activation function named Mish has better results than ReLu with deep learning tasks [35]. Mathematical representation of ReLu:

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

On the other hand, Softmax activation function is used commonly in the output layer after the fully connected layer for classification. It is highly suitable for multiclass classification [23]. Softmax activation function transforms the values in the output layer into meaningful probability distributions for output classes, which can be used for predictions. Probability distributions are normalized in a range of 0 to 1, and sum of all the probability distributions adds up to 1. Mathematical representation of Softmax function:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \dots, K \text{ and } z = (z_1, \dots, z_K) \in \mathbb{R}^K \tag{6}$$

It is to be noted that the use of linear "Support Vector Machines" instead of "Softmax" leads to better results for the classification [36]. However, in the experiments, "Softmax" is preferred due to availability in used frameworks.

The rest of the activation functions are not covered since they are not used in the experiments of the thesis.

2.4.3. Information Propagation

The components of the CNN are covered in the previous section. In this section, processes of the computations done through the network are explained. The operations can be classified under two categories depending on the direction of the process. The operations conducted from the input layer to the output layer can be classified under forward propagation, and the processes conducted in the reverse order can be classified under backpropagation.

2.4.3.1. Forward propagation

In the forward propagation, the operations are conducted in the direction from the input layer to the output layer (as described with the blue arrow in Figure 2.2). For the first operation, iteration number 1, the predictions are made according to the initial weights and biases. And for the subsequent iterations, the predictions are made according to the updated weights and biases.

2.4.3.2. Backpropagation

The backpropagation operations are conducted from the weight matrix between the output layer and the fully connected layer, which is not necessarily fully connected, to the weights of filters in the first convolutional layer (as described with the red arrow in Figure 2.2). During

the backpropagation, weights for convolutional layers and also the fully connected layer weights are updated. There are no parameters to be updated for pooling layers.

At the core of the backpropagation operations lies the backpropagation algorithm. The first studies on the backpropagation algorithm started with the studies of Seppo Linnainmaa [37], which is followed by the discussions of applications of backpropagation on neural networks [38]. The concrete steps for the application of backpropagation on neural networks were made in 1986 [19]. The studies have shown that the backpropagation algorithm can be used to train neural networks.

In its basic form, the backpropagation algorithm aims to optimize the network's weights to minimize the error between the output of the predictions and the target output. For the optimization process, various methods are used, such as gradient descent [39] and variations of gradient descent. Primary variations of gradient descent include Batch Gradient Descent, Mini Batch Gradient Descent [40], Stochastic Gradient Descent (SGD) [41]. There are also different schemes available for the tunning of the parameters (such as step size) for optimization, such as AdaGrad (Adaptive Gradient) [42], Adam (Adaptive Moment Estimation) [43], RMSprop (Root Mean Square Propagation) [44], etc.

In the experiments, as a step-size optimizer, Adam optimizer is used. The crucial difference between the Adam algorithm and other methods is that it also uses the cumulative history of gradients. It is a frequently used optimizer for neural networks. Furthermore, the TensorFlow framework uses a learning rate of 0.001 for the Adam optimizer as default. The Adam algorithm updates the weights according to equations :

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t \tag{7}$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2 \tag{8}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{and} \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{9}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} * \hat{m}_t \tag{10}$$

where m_t represents the first-moment vector, v_t represents the seconds moment vector, w_t represents the weight matrix, w_{t+1} represents the updated weight matrix, and ∇w_t is the gradient of weight matrix. The parameters β_1 and β_2 are 0.9 and 0.999 by defeault, respectively. The constant ϵ is set to 10^{-8}, which is used to avoid having zero in the denominator. η is the step size and, as stated, set to 0.001.

The errors in the predictions are determined using loss functions, and the loss is iteratively minimized by taking gradient descent steps potentially optimized by step-size optimizers like Adam. The process is the key point for achieving low error. However, it is the primary reason for high training times. There are several loss functions available. Since the tasks in the

experiments are classification and in the output layer, the Softmax activation function is used, the outputs of the predictions are probability distribution. Therefore, the categorical cross-entropy loss function is chosen. The loss value can be calculated using the formula:

$$L_i = -\sum_j y_{i,j} \log(\hat{y}_{i,j}) \tag{11}$$

where $\hat{y}$ is the predicted distribution, y is the actual distribution, i represents the sample position and j is the label index.

The derivation of the backpropagation algorithm is available in the literature [45].

The backpropagation algorithm is replaced with a non-iterative pseudo inversion analytical method in the ELM algorithm, which will be explained in the next section.

2.5. Extreme Learning Machines

The conventional feedforward neural networks are trained using gradient-based learning algorithms, which are slow. In the complex forms of neural networks, including a high number of parameters, computational complexity is huge. Furthermore, the parameters are updated iteratively, and these learning algorithms suffer local minima problems [46].

Extreme Learning Machine (ELM) is an algorithm proposed to provide a better solution for the performance of feedforward neural networks. According to the ELM algorithm, the hidden neurons can be randomly generated without being tuned, and the output weights are analytically determined. It is distinguished from standard feedforward neural networks by not including gradient-based learning algorithm. Furthermore, the algorithm might have a better generalization performance than the backpropagation algorithm [46]. Non-iterative determination of output weights reduces learning time and contributes to efficiency. However, due to its structure, the algorithm solely may not perform well with tasks including feature learning [47].

The algorithm is first introduced by Guang-Bin Huang et al. in 2004 [5]. As Huang, G.-B. explains, it is inspired by biological learning systems and proposed to overcome the issues faced by backpropagation learning algorithms. They conjecture that the brain systems have random neurons with parameters independent of the environment [48]. “Extreme” in the name refers to moving beyond the conventional learning techniques [48]. From its discovery, the algorithm has been used in combination with several methods and researched by Huang himself a lot as well. ELM variants include: “Voting based extreme learning machine” [49], “On-Line Sequential Extreme Learning Machine” [50], “Evolutionary extreme learning machine” [51], etc. There are several speculations for the origins of ELMs, suggesting that the “Random Vector Functional Link (RVFL) network” [52], “Feed Forward Neural Networks with Random Weights” [53], “A weight initialization method for improving training speed in feedforward neural network” [54], etc. are the origins of ELM. However, Guang-Bin Huang

states that this is not the case [55]. In 2006, the implementation of the algorithm for training Single Layer Feed-Forward Neural Networks (SLFNs) was published [46]. To achieve accuracy improvements, in 2016, the implementation of the algorithm for Two-Hidden Layer Neural Networks (TLFNs) is proposed [56].

The advantages of ELMs over CNNs can be compromised into several matters. Trainings are accomplished in a single iteration and training times are significantly less [12]. The necessity of big amount of data may be seen as a constraint for CNNs.

Extreme Learning Machines are an essential part of this thesis and are promising for future research. The performance benefits of it against backpropagation are attractive from a scientific point of view. In the following sections, the algorithms for Single-Hidden-Layer Feed-Forward Networks and Two-Hidden-Layer Feed-Forward Networks are explained. Both methods are used in the experiments.

2.5.1. Single-Hidden Layer Extreme Learning Machine

The ELM algorithm for Single-Hidden Layer Feed-Forward Neural Network was introduced in 2006 and later extended to generalized Single-Hidden Layer Feed-Forward Networks [46]. The input layer and hidden layer are fully connected, and the weights between the two layers are randomly initialized. The structure of a Single-Hidden Layer Feed-Forward Neural Network can be seen in Figure 2.9.

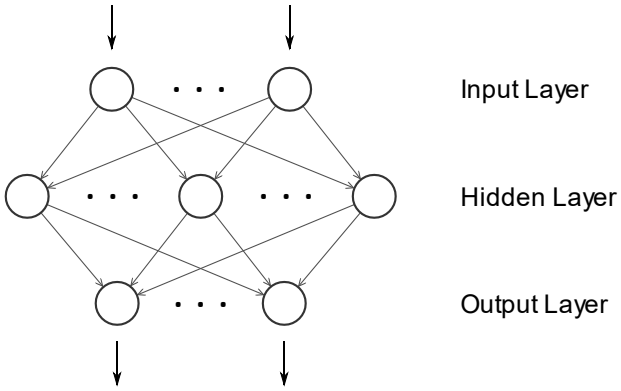

Figure 2.9 Single-Hidden Layer Feed-Forward Network

The input matrix for the input layer is given as:

$$\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n]^T \tag{12}$$

And the weight matrix between input and hidden layer $\boldsymbol{W}$, and bias vector $\boldsymbol{\Theta}$ is given as:

$$\boldsymbol{W} = \begin{bmatrix} \boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_m \end{bmatrix}^T \text{ and } \boldsymbol{\Theta} = \begin{bmatrix} \boldsymbol{\Theta}_1 \\ \vdots \\ \boldsymbol{\Theta}_m \end{bmatrix} \tag{13}$$

The output values for the input layer and randomly initialized weights and biases are stored in the matrix $\boldsymbol{H}$. During the propagation, activation function is applied. The hidden layer output matrix $\boldsymbol{H}$ can be represented as:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{h}(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{h}(\boldsymbol{x}_n) \end{bmatrix} = \begin{bmatrix} \sigma(\boldsymbol{w}_1\boldsymbol{x}_1 + \boldsymbol{\Theta}_1) & \cdots & \sigma(\boldsymbol{w}_m\boldsymbol{x}_1 + \boldsymbol{\Theta}_m) \\ \vdots & \ddots & \vdots \\ \sigma(\boldsymbol{w}_1\boldsymbol{x}_n + \boldsymbol{\Theta}_1) & \cdots & \sigma(\boldsymbol{w}_m\boldsymbol{x}_n + \boldsymbol{\Theta}_m) \end{bmatrix}_{n\times m} \tag{14}$$

Where σ is the non-linear activation function, m is the number of neurons in the hidden layer, n is the number of inputs in input layer, and $\boldsymbol{h}(\boldsymbol{x})$ is the hidden layer output mapping, which can be represented as:

$$\boldsymbol{h}(\boldsymbol{x}) = [h_1(\boldsymbol{x}), \dots, h_m(\boldsymbol{x})] \tag{15}$$

The hidden layer and the output layer are fully connected as well, and the weights between the layers are represented by the weight matrix $\boldsymbol{\beta}$. Weight matrix $\boldsymbol{\beta}$ and training output $\boldsymbol{T}$ is given as:

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{t}_1^T \\ \vdots \\ \boldsymbol{t}_n^T \end{bmatrix}_{n\times l} \quad and \quad \boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \vdots \\ \boldsymbol{\beta}_m \end{bmatrix}^T_{m\times l} \tag{16}$$

l is the number of outputs. We are interested in minimizing the equation below. It is minimized when the weight matrix $\boldsymbol{\beta}$ is chosen optimally:

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{H}\boldsymbol{\beta} - \boldsymbol{T}\| \tag{17}$$

According to the ELM algorithm, the output weight matrix $\boldsymbol{\beta}$ is not pre-determined and not random initialized. Least-squares solution for the equation 17, $\widehat{\boldsymbol{\beta}}$, is determined with the formula:

$$\widehat{\boldsymbol{\beta}} = \boldsymbol{H}^{\dagger}\,\boldsymbol{T} \tag{18}$$

where † denotes the Moore-Penrose generalized inverse, and the solution is unique. Moore-Penrose generalized inverse of $\boldsymbol{H}$ is calculated with the formulas:

$\boldsymbol{H}^{\dagger} = (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T$ in case for $\boldsymbol{H}_{nxm}$, $m < n$ or $\boldsymbol{H}^{\dagger} = \boldsymbol{H}^T(\boldsymbol{H}^T\boldsymbol{H})^{-1}$ in case for $\boldsymbol{H}_{nxm}$, $m > n$

Therefore, the output weight matrix is analytically determined. The output function of the SLFN for the j^{th} input can be represented as:

$$F(\boldsymbol{x}_j) = \sum_{i=1}^{m} \boldsymbol{\beta}_i\, h_i(\boldsymbol{x}_j) = \sum_{i=1}^{m} \boldsymbol{\beta}_i\, \sigma(\boldsymbol{w}_i\boldsymbol{x}_j + \boldsymbol{\Theta}_i), \qquad j = 1, \dots, n \tag{19}$$

m is the number of neurons in the hidden layer, n is the number of inputs in input layer.

2.5.2. Two-Hidden-Layer Extreme Learning Machine (TELM)

The motivation behind the extension of the previous section is to achieve better accuracy. In the TELM algorithm, a hidden layer is added to single-hidden-layer ELM architecture [56]. The idea is based on the results regarding the performance of two-hidden-layer feed-forward neural networks (TLFNs) [57]. The experimental results suggest that the TELM performs better in accuracy than Single-Hidden-Layer ELM as well as other multilayer ELM variants [56]. TELM algorithm is applied to the TLFNs with the structure in Figure 2.10.

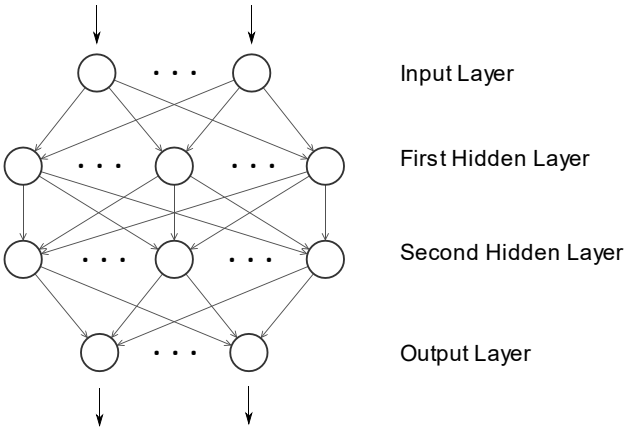

Figure 2.10 Two-Hidden Layer Feed-Forward Network

The following algorithm is not the standard TELM algorithm [56]. The algorithm is based on inverting the output matrices instead of distributing the input space. Same operations for the Single-Hidden-Layer can be followed until the second hidden layer is reached. It means that $\boldsymbol{\beta}$, in this case weight matrix between first hidden layer and second hidden layer, can be determined using the equations 14, 17 and 18. The training output of the first hidden layer can be considered as the input for the second hidden layer. Following equation shows the determination of the expected output for the second hidden layer:

$$\widehat{\boldsymbol{H}}_2 = \boldsymbol{T}\boldsymbol{\beta}^{\dagger} \tag{20}$$

where $\boldsymbol{T}$ is the training output and $\boldsymbol{\beta}^{\dagger}$ is the Moore-Penrose generalized inverse of the weight matrix between first hidden layer and second hidden layer. Since $\widehat{\boldsymbol{H}}_2$ is determined, the weights matrix between the first and second hidden layer can be calculated with the equation:

$$\boldsymbol{W}_{HE} = \sigma^{-1}\,(\boldsymbol{H}_2)\boldsymbol{H}_E^{\dagger} \tag{21}$$

Where $\boldsymbol{H}_E = [1 \quad \boldsymbol{H}]^T$, $\boldsymbol{H}$ is the first hidden layer output, as in equation 14. $\boldsymbol{H}_E^{\dagger}$ is the Moore-Penrose generalized inverse of $\boldsymbol{H}_E$, and $\boldsymbol{W}_{HE} = [\boldsymbol{\Theta}_1 \quad \boldsymbol{W}_H]$. The $\boldsymbol{\Theta}_1$ and $\boldsymbol{W}_H$ represent the bias matrix and the weight matrix between the first and second hidden layer, respectively. σ^{-1}

represents the inverse of activation function for the second hidden layer. The actual output for the second hidden layer is determined with the equation:

$$\boldsymbol{H}_2 = \sigma(\boldsymbol{W}_{HE}\boldsymbol{H}_E) \tag{22}$$

Thereby, the weight matrix between second hidden layer and output layer can be calculated with the equation:

$$\boldsymbol{\beta}_{new} = \boldsymbol{H}_2^{\dagger}\boldsymbol{T} \tag{23}$$

Furthermore, the output of the network can be calculated with the equation:

$$Y = \sigma(\boldsymbol{W}_H\sigma(\boldsymbol{W}\boldsymbol{X} + \boldsymbol{\Theta}) + \boldsymbol{\Theta}_1)\boldsymbol{\beta}_{new} \tag{24}$$

2.6. Raspberry Pi

Raspberry Pi is a credit-card-sized Single Board Computer (SBC) developed in the United Kingdom by the Raspberry Pi Foundation. The Foundation was founded in 2009, and the development of Raspberry Pi's was for educational purposes [58]. The first generation of Raspberry Pi (Raspberry Pi Model B) was released in February 2012.

Since the release of the first Raspberry Pi's, they have started to be used for important applications and become attractive because of its affordable price and relatively small size. One of the most important applications of the Raspberry Pi is Internet of Things (IoT) [9]. The CPU of the Raspberry Pi is cheap, powerful, and does not consume much power. Furthermore, Raspberry Pi supports wireless connection and a wide range of input and output peripherals. Through General-Purpose Input/Output (GPIO), connection of external electronics is possible [59]. Therefore, it is a valuable candidate for interfacing with multiple devices and applications and suitable for IoT applications [60]. Considering the high potential for the development of applications in IoT [61], further involvement of the Raspberry Pi is inevitable.

Moreover, Raspberry Pi has similarities with other mobile devices from the performance and architectural perspective. Thus, Raspberry Pi contributes to the development of other concepts as well. Those broad and developing application areas and the increasing popularity of Raspberry Pi make it an attractive concept to research.

An SBC is a fully functional computer system. It has microprocessor(s), memory, input/out, and other components [58]. A circuit board and components of a Raspberry Pi 4B can be seen

below. As indicated in Figure 2.11, through the GPIO pins, external electronic components can be connected [59].

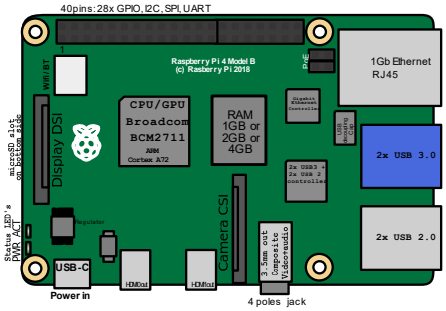

Figure 2.11 Raspberry Pi 4B [1]

There are other SBC's such as Galileo Ordroid. Banana Pi, Cubieboard, etc. [62], which are mostly used on single-sided Printed Circuit Board (PCB). Since all the components are integrated on a single PCB, it is not possible to upgrade the hardware. Furthermore, SBC's are System on a Chip (SoC). [58] SoC integrates on-chip microprocessor/microcontroller with components like RAM, GPU, interface cores (PCI, USB, etc.) on a single chip [63]. SoCs are also preferred for smartphones and tablets in today's market due to their cost efficiency, size, and low power consumption.

The most powerful Raspberry Pi available in the market is Raspberry Pi 4B. It uses BCM2711 (by Broadcom) as an SoC. Moreover, Raspberry Pi's use Acorn RISC Machine (ARM) as Instruction Set Architecture (ISA). An ISA is an abstract model of a computer. The ARM architecture is Reduced Instruction Set Computer (RISC) and targets low energy consumption; thus, it is highly suitable for mobile systems [64]. Today ARM architecture is prevalent in mobile systems, and it started to be popular among computers. As a popular example, recently introduced (November 10, 2020) Apple M1 chip is the first ARM-based chip as a CPU for Mac computers [2].

Considering its simpler instruction set architecture and low power demand, it becomes attractive compared to Complex Instruction Set Architectures (CISC), which is currently used by most computers [65]. On the one hand, there are some compatibility issues with the software designed for computers. On the other, it is widely used by mobile devices. Furthermore, the compatibility issues can be solved by developers [65]. There are also new approaches about ISA for Neural Network accelerators, which is crucial considering the not-energy-efficient execution of Neural Networks on CPUs [66].

With developments on the side of the mobile platform and increasing popularity of them makes mobile platforms attractive. Raspberry Pi is becoming one of the promising and developer-friendly mobile platforms gives an important glimpse for other platforms. The motivation behind the first development of Raspberry Pi might differ from today's application areas. The reason for that is the increasing importance of mobility.

[1] „RaspberryPi 4 Model B" by Jstrom99 is licenced under CC BY-SA 4.0
[2] Bonov, P. *The Apple M1 is the first ARM-based chipset for Macs with the fastest CPU cores and top iGPU*. 2020, November 10; Available from: https://www.gsmarena.com/.

2.6.1. Methods to improve performance of Raspberry Pi

Over the years, the architecture of the Raspberry Pi has changed, and the computational power of it has increased drastically. However, training a neural network requires a vast amount of computational power. Therefore, there are methods developed to increase performance on Raspberry Pi. For instance, the computations (in the matter of deep learning) started to be used on GPU rather than CPU. The GPU is a suitable medium for deep learning tasks since it can utilize the tasks in parallel. Moreover, GPUs are useful for matrix multiplication and convolution, which is beneficial for deep learning tasks [67].

Since the Raspberry Pi have a relatively weak GPU (Broadcom VideoCore VI @ 500 MHz), the models are trained on its CPU. Moreover, the framework used for the experiments, Tensorflow, does not have GPU enabled version for Raspberry Pi. It is to be noted that NVIDIA Jetson's peak GPU performance is above the Raspberry Pi, making it more suitable for deep learning tasks [68]. Nevertheless, considering its high price, it is not feasible for practical applications.

In the following sections, methods to improve the performance of Raspberry Pi are introduced.

2.6.1.1. Overclocking

In order to meet the demands of today's technology world, performance improvements for CPUs have been made. Moreover, there are methods integrated into CPU architectures to increase the performance of CPUs at hand. One of these methods is dynamic overclocking. Since Raspberry Pi has limited space, the method is beneficial to improve performance without any hardware enhancement.

In dynamic overclocking, the maximum CPU clock speed is increased. Instead of constantly holding the CPU clock speed on the same level, dynamic overclocking adjusts CPU clock speed according to need. In the case of maximum load, the CPU clock speed will be maximized. In the experiments of this thesis, all four cores of the CPU of Raspberry Pi 4b were involved during the training operations, and the cores were operating at full CPU clock speed.

According to experimental results [3], with the dynamic overclocking method, Raspberry Pi 4 can reach a CPU clock speed of 2.147 GHz. Without the method, the maximum CPU clock speed is 1.5 GHz. However, in the experiments, the CPU is overclocked to a maximum CPU clock speed of 2 GHz. Since above this value, there are considerable damage risks for the hardware.

[3] Hattersley, L. *How to overclock Raspberry Pi 4*. 2019; Available from: https://magpi.raspberrypi.org/.

Even under the 2.147 GHz CPU clock speed, the method has some risks. With the frequent application of the method, lifespan may be reduced. In the overclocked state, the components of the Raspberry Pi heats up more and are exposed to higher voltage compared to the normal state, which can immediately damage the hardware.

2.6.1.2. ZRAM Method

ZRAM method, formerly called "compcache" [4], is used to create RAM-based block devices (named zram), acting as a swap disk and allow more space for memory. Since 2014, January 14, it is part of the official Linux kernel. The created swap memory will be on the storage device of the Raspberry Pi, which is the microSD card. It serves as additional memory space, when necessary, with an effective compression method. There are several options available as a compression method for the ZRAM method. In the experiments for this thesis, the LZ4 [5] compression algorithm is used.

In consequence of using the method, there is only a maximum of 1% increase in CPU usage due to compressing and decompressing operations. It can be concluded that the method does not have a negative effect on the performance of the CPU.

The method does not only contribute to the performance, it also enables the execution of programs, which would fail due to memory allocation issues [6]. Deep learning tasks use lots of memory since there are many parameters, input data, etc., to be stored. In the experiments for this thesis, there were memory issues when executing the programs.

The method can be applied using the script [7] installation and configuration.

2.6.2. Risks for training models on Raspberry Pi

With the developments over the years, Raspberry Pi's became more powerful. However, at the same time, there are several hardware issues. Newer versions consume more energy and generate more heat. For example, BCM2711(used by Raspberry Pi 4) requires 6.67-7.20-Watt (depending on the version of the firmware) on the load mode supply provided by the USB-C port. Here, load mode indicates the computationally demanding mode. The former version of it (Raspberry Pi 3B+) consumes approximately 5.77 Watt on the load mode [69].
Due to prolonged usage of the Raspberry Pi, there may be some problems. The problems include overheating, and thus thermal throttling etc. Even though, due to low power consumption of Raspberry Pi, it does not dissipate much heat but for the tasks with high workload, thermal throttling should be considered.

[4] Google Code. *Compcache*. 2009; Available from: https://code.google.com/archive/p/compcache/.
[5] Collet, Y. *LZ4 - Extremely fast compression.* 2020; Available from: https://github.com/lz4
[6] Gupta, Nitin. *Compcache: in-memory compressed swapping*. 2009, May 26; Available from: LWN.net.
[7] Scott, B. *zram-swap*. 2019; Available from https:/github.com/foundObjects/zram-swap. Licenced under MIT License

Thermal throttling is a power management technique to reduce the performance of CPU in the presence of excessive temperatures that might cause malfunction of the device. Overheating and thus thermal throttling affect the performance of Raspberry Pi on tasks. Raspberry Pi's processor is designed to withstand temperatures in the range of -40°C to 85°C. The effects of thermal throttling occur above 80°C. Under the constant exposure to 80°C, after approximately 65 seconds, the clock speed drops from 1.5 GHz to 1 GHz. And after approximately 600 seconds, Raspberry Pi stops working. Performance of the Raspberry Pi 4's CPU over time on the load mode can be seen from the below Figure 2.12.

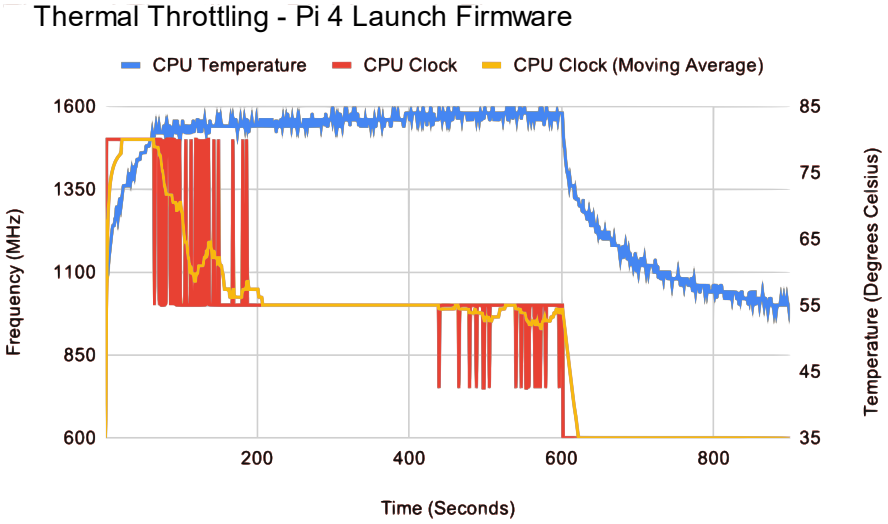

Figure 2.12 Raspberry Pi 4 CPU Performance [8]

[8] Halfacree, G., *Thermal Testing Raspberry Pi 4*, in *The MagPi Issue 88*. 2019, Raspberry Pi Press. p. 66-74.

3. Experimental Setting

In the experimental setting and methods sections, the methodology used for the experiments and development of the codes are explained. At the beginning of this section, an introduction to libraries, frameworks, and datasets used for development and essential parts of the research are given. Subsequently, the methodology used to develop the codes and configurations for the Raspberry Pi and the computer is explained.

Python programming language was used for the implementation of the model into Raspberry Pi. The Linux distributions for the Raspberry Pi come with Python pre-loaded, and programming can be done smoothly on Raspberry Pi by using the IDLE (Integrated Development and Learning Environment). For the development, Python 3.x is preferred against Python 2.x since most of the new projects are using Python 3, contributing significantly to available documentation.

3.1. Libraries and Frameworks

The popularity of Python grows faster than any other programming language. Furthermore, in the topic of machine learning, it is considered to be the most preferred language [70]. Parallel to the growth of Python, there have been frameworks and libraries developed. The usage of frameworks and libraries is beneficial for the efficiency of development. Some of them find a wide range of applications, and some are designed for specific purposes. The following frameworks and libraries were used to implement CNN, ELM, and hybrid models.

3.1.1. NumPy

NumPy is a mathematical library to perform scientific computations fast. The library uses optimizing compilers and provides speed advantages. Moreover, it allows low-level operations and provides packages for linear algebra, random number generation, etc. [71].

3.1.2. SciPy

The initial ideas for the SciPy emerged from the need for a complementary package on top of the Numeric array package for more complex scientific tools. There were scientific packages designed for various purposes, and in 2001, they were gathered under a single library by Enthought Inc. [72].

Both NumPy and SciPy are Python libraries designed for scientific operations. Compared to SciPy, NumPy is faster for basic operations and not suitable for complex tasks involving statistics and data manipulation. SciPy and NumPy have lots of overlapping points and compatible with each other. Therefore, they are frequently applicable together to deep learning, where they are assigned for different tasks.

3.1.3. Scikit-learn

Scikit-learn is a Python library used mainly for machine learning tasks. It provides simple and efficient implementations of predictive statistical analysis tools. Furthermore, it depends only on NumPy and SciPy, which are fundamental scientific libraries. Its license conditions (BSD license), open-source community, smooth installation, and portability make it popular and convenient for commercial and academic settings [73].

3.1.4. TensorFlow

TensorFlow is a framework designed by the Google Brain team for creating Machine Learning models, and in particular, Deep Learning models. TensorFlow uses dataflow graphs to represent computations and accept data in the form of tensors (multi-dimensional arrays) [74]. It offers developmental aids for neural networks and optimized models; thus, allows easier coding. TensorFlow supports the development of models on CPU and GPU of computers. In the case of Raspberry Pi, TensorFlow currently allows only developments for deployments on CPU.

There is a lightweight version of TensorFlow named TensorFlow Lite. TensorFlow Lite is designed for deploying machine learning models on mobile and IoT devices. It does not support on-device training yet; however, the TensorFlow team is working on the subject [9]. Therefore, it is not plausible to use TensorFlow Lite as a framework for the experiments.

3.1.5. Keras

Keras is a high-level deep learning API written in Python, running on top of TensorFlow and Theano (a Python library for deep learning). [75]. TensorFlow provides both high-level and low-level APIs, while Keras provides only high-level APIs. Low-level APIs allow developers to change code in detail, whereas high-level APIs provide ready-to-use functions, customizable models, and more functionality [76].

3.1.6. Others

Other libraries were also used for the development, such as "tqdm" [10], to keep track of the training's progress. Alternatively, Matplotlib was used to make graphics for the interpretation of the result. Nevertheless, since they do not influence the implementation of the models, they are not explained here explicitly.

[9] TensorFlow. *TensorFlow Lite*. 2020, January 28; Available from: https://www.tensorflow.org/lite
[10] da Costa-Luis, C. *tqdm*. 2018; Available from https://github.com/tqdm/tqdm. Licenced under MIT License

3.2. Datasets

Datasets are one of the essential components of training and testing of the models. Image classification task was used for the experiments consisting of assigning a label to an input image from a fixed set of categories [11]. Furthermore, image classification is a fundamental task in the field of computer vision. Therefore, image classification datasets are of great importance. MNIST and Fashion MNIST datasets are popular datasets for image classification. Both are open datasets and suitable for scientific research purposes.

In the experiments, both models and their combination were trained on the MNIST and Fashion MNIST datasets, and the results were recorded for different parameters. The Fashion MNIST is comparatively a difficult dataset [12].

3.2.1. MNIST

The MNIST database consists of 60,000 training samples and 10,000 test samples. Every sample is a 28x28 grayscale image of handwritten digits, ranging from 0 to 9 [28]. In other words, the images are associated with labels from 10 categories. The humans have an approximately 0.2% error rate and 97.5-98% accuracy for the task [77]. The lowest reported error rate for the MNIST dataset to date is 0.16% when non-human systems are used; thus, with at least 99.84% accuracy [78], which is beyond human recognition capabilities. Example images for MNIST dataset are shown in Figure 3.1.

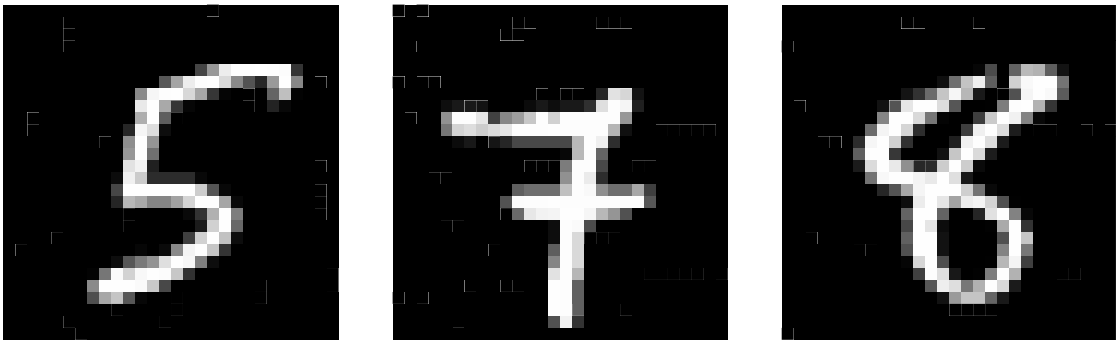

Figure 3.1 Images from MNIST Dataset. The images represent, three of the samples from MNIST dataset. The images are from left to right labelled as: “five”, “seven”, and “eight”.

3.2.2. Fashion MNIST

Fashion-MNIST also consists of 60,000 training samples and 10,000 test samples. Each sample is a 28x28 grayscale article image of a clothing object from Zalando’s database and consists of 784 pixels. The pixel values are integer numbers differentiating between 0 and 255. These values represent the darkness of each pixel on the image. Human performance for

[11] Karpathy, A. *CS231n Convolutional Neural Networks for Visual Recognition*. 2016; Available from: https://cs231n.github.io.
[12] Zalando Research. *fashion-mnist*. 2017; Available from https://github.com/zalandoresearch/fashion-mnist. Licenced under MIT License

the dataset is approximately 83.5% accuracy [33]. The lowest reported error rate for the Fashion MNIST dataset is 3.09% for artificial systems, with an accuracy of 96.91% [79]. The difference between human recognition performance and neural network performance can be readily examined using the Fashion MNIST since it is a much more challenging dataset. Example images for Fashion MNIST can be seen in Figure 3.2.

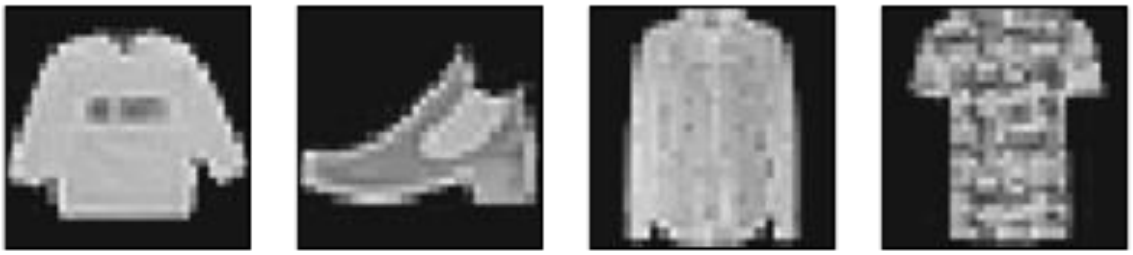

Figure 3.2 Images from Fashion MNIST Dataset. The four images are from Fashion MNIST dataset. They are from left to right labelled as follows: “pullover”, “ankle boot”, “shirt”, and “t-shirt”. There are 6 more labels for the images in Fashion MNIST.

3.3. Raspberry Pi and Computer Configurations

In this section, configurations for the computer and Raspberry Pi are introduced. Each platform is discussed in separate sections.

3.3.1. Raspberry Pi Configurations

For the experiments with the Raspberry Pi’s, Raspberry Pi 4 Model B is used. It is the newest model available in the market and powerful compared to other models (Figure 3.3).

Product	SoC	Speed	RAM	Wireless
Raspberry Pi Model B+	**BCM2835**	**700MHz**	**512MB**	**No**
Raspberry Pi 2 Model B	**BCM2836/7**	**900MHz**	**1GB**	**No**
Raspberry Pi 3 Model B+	**BCM2837B0**	**1400MHz**	**1GB**	**802.11ac/n**
Raspberry Pi 4 Model B	**BCM2711**	**1500MHz**	**2GB**	**802.11ac/n**
Raspberry Pi Zero W	**BCM2835**	**1000MHz**	**512MB**	**802.11n**

Figure 3.3 Chart for Raspberry Pi Models[13]

Instead of using the Graphical User Interface (GUI) of Raspberry Pi and generate extra load on the device, the Raspberry Pi is controlled over an SSH server. Moreover, as an external storage unit Micro SDHC card (16 GB, compatible with Raspberry Pi OS), and for writing the operating system on Raspberry Pi, Raspberry Pi Imager [14] is used.

[13] Raspberry Pi Foundation. *Frequently asked questions*; Available from: https://www.raspberrypi.org
[14] Hollingworth, G. *Introducing Raspberry Pi Imager, our new imaging utility*. 2020; Available from: https://www.raspberrypi.org.

Raspberry Pi OS, formerly named Raspbian, was the operating system used. It is a Debian-based operating system officially provided by the Raspberry Pi Foundation and highly optimized for Raspberry Pi. Additionally, it is free and open-source. The Raspberry Pi OS has two versions available: Raspberry Pi OS Full and Raspberry Pi OS Lite. The statistics on the frequency of the usage of them can be found on the internet [15]. Raspberry Pi OS Lite does not have the libraries, video drivers, and unnecessary pre-downloaded software and does not cause extra load on CPU and RAM, which will be under substantial load during the training of neural networks. Moreover, it is the preferred operating system for a "headless" Raspberry Pi setup. Thus, it was selected.

The Secure Shell (SSH) Protocol [80] is a protocol for remote network services over a network. OpenSSH [16] 7.7 is used during the experiments. In order to join Raspberry Pi in the SSH server, the necessary files are transferred into Raspberry Pi. Since November 2016 release, Raspberry Pi OS has the SSH server disabled by default due to security reasons [17]. Another useful tool for working remotely and file sharing with Raspberry Pi is WinSCP software, which Raspberry Foundation recommends, was also used.

In the next step, necessary Python packages had to be installed. For the installment and management of packages, pip3 [18] 18.1 was used. The necessary packages included h5py (2.10.0), NumPy (1.20.2), pandas (0.23.3), sci-kit learn (0.20.2), sci-py (1.4.1), TensorFlow (2.2.0), and Keras (2.4.3).

In order to get TensorFlow work on Raspberry Pi, the following packages should be additionally installed: libhdf5-dev, libc-ares-dev, libeigen3-dev, libatlas-base-dev, libopenblas-dev, libblas-dev, openmpi-bin, libopenmpi-dev, liblapack-dev, cython, keras_applications==1.0.8 --no-deps, keras_preprocessing==1.1.0 --no-deps

For the implementations on Raspberry Pi, TensorFlow 2.0.0 is used because there were some problems between the h5py version 2.10.0 package dependency and TensorFlow 2.4.0 (latest version available). Considering stability, TensorFlow 2.2.0 and h5py 2.10.0 are used.

A comprehensive guide for the configuration of TensorFlow on Raspberry Pi can be found online [19].

Furthermore, to avoid any problem due to heating, a cooling system was applied to Raspberry Pi. The cooling system consists of a heatsink and a cooling fan. Overheating occurs in the areas near the SoC (red area in Figure 3.4). Therefore, the cooling system was applied over that area. In the experiments, the SoC was used for the training of the models.

[15] Raspberry Pi Foundation. *RPi Imager Stats*; Available from: https://rpi-imager-stats.raspberrypi.org
[16] Open SSH. OpenSSH Project History; Available from: https://www.openssh.com/
[17] Raspberry Pi Foundation. *SSH (Secure Shell)*; Available from: https://www.raspberrypi.org
[18] The Python Packaging Authority. *pip*; Available from: https://pip.pypa.io/
[19] Q-engineering. *Install TensorFlow 2.2.0 on Raspberry Pi 4*; Available from: https://qengineering.eu/

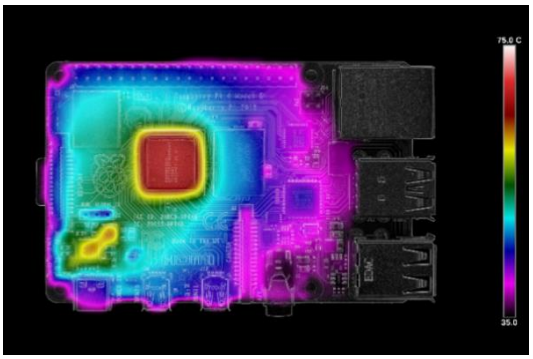

Figure 3.4 Thermal Image of Raspberry Pi 4 in Load Mode [20]

Since previous experiments confirm [69] that Raspberry Pi 4 performs better and heats up slowly when placed vertically, vertical orientation was preferred for the experiments. Furthermore, the Raspberry Pi was not placed inside of its case for optimal ventilation.

Another critical point was the power supply. Typically, a power supply with the specifications of 5.0V/2.0A is sufficient for operating Raspberry Pi and cooling fan. However, there was a decline in performance during the experiments with overclocked Raspberry Pi using a 5.0V/2.0A power supply. Therefore, the experiments had to be repeated with a sufficient power supply for overclocking operation (an adapter with a 65W power supply was used).

During the training of the models, there were some memory problems. During the implementation process, when memory problems were encountered, the ZRAM method was used. To be precise, 4 GB of swap space was created with the ZRAM method.

Moreover, performance of the devices is monitored during the experiments. Considering that monitoring is vital for preserving the stability of devices during the experiments, the following parameters of device performance were monitored: CPU clock speed, percentage of RAM and CPU usage, and active cores of the CPU.

The clock speed of the CPU of Raspberry Pi is monitored using the command line utility "vcgencmd" [21] with the command:

```
watch -n 1 vcgencmd measure_clock arm
```

with that tool, Raspberry Pi's real-time CPU clock speed can be tracked on the command line. The monitoring software uses only approximately 1.5% of the CPU resource.

Active cores of the CPU, total processor, and RAM utilization can be observed with the htop [22] tool, which uses less than 1% of the CPU.

[20] Halfacree, G., *Thermal Testing Raspberry Pi 4*, in *The MagPi Issue 88*. 2019, Raspberry Pi Press. p. 66-74.
[21] Raspberry Pi Foundation. *VCGENCMD*; Available from: https://www.raspberrypi.org
[22] Muhammad, H. *htop*. 2004; Available from https://github.com/htop-dev/htop. Licenced under GNU General Public License v2.0

3.3.2. Computer configurations

Computer experiments were conducted on a laptop with Intel Core i7 (2nd generation) 2.2 GHz processor with four cores and eight threads. It has 8 GB of RAM and an Nvidia GeForce GT555M 1 GB graphics card. However, the GPU was not used in the experiments. The computer runs on Ubuntu 20.04.2 LTS (64 bit).

There was no special procedure to be followed to install the packages. Everything could be done over pip3 in the terminal. The necessary core packages and respective versions were as follows: h5py (2.10.0), numpy (1.19.5), pandas (1.2.4), sci-kit learn (0.24.1), sci-py (1.6.2), TensorFlow (2.4.1), and Keras (2.4.3).

The experiments were run on the CPU of the computer to provide a fair comparison. Since the computer's operating system was also a Linux distribution, the same monitoring tools used for Raspberry Pi could be used for the computer as well. However, since “vcgencmd” tool is included only in Raspberry Pi OS, this tool was not used in computer experiments. Moreover, for the optimal conditions of operations, a cooling pad was used.

The experiments on the computer were also conducted over an SSH server to minimize load on CPU and RAM.

4. Methods

Experimenting with the models is the fundamental stage of this thesis. The experiments consist of two sections. In the first section CNN, ELM, and hybrid models were trained for different parameters (wide range of combinations), and the experiments were conducted on the computer and Raspberry Pi 4B without overclocking.

Furthermore, both datasets were used. The results for training time, training accuracy, and test accuracy were recorded. The performance of the models with different parameters was compared regarding the training time and test accuracy, and the parameters with superior performance were qualified for the second phase of the experiments. The qualifying parameters can be seen in Figure 4.1 under "Tested parameter combinations".

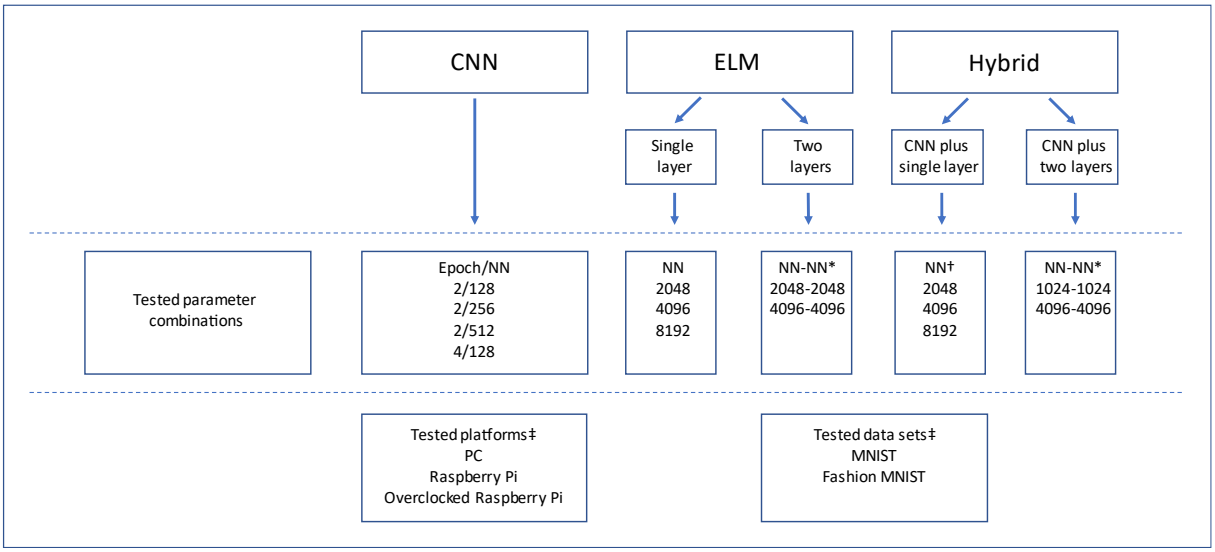

Figure 4.1 Summary of thesis experiments. The meaning of abbreviations are as follows: NN, number of neurons; CNN, convolutional neural networks; ELM, extreme learning machine. The meaning of the symbols are: *number of neurons for each layer, †number of neurons for the latest step (ELM); ‡all experiments were conducted for all three platforms and two data sets.

In the second stage of the experiments, all experiments were conducted for both platforms and datasets. However, in the second stage, the experiments were also conducted on the overclocked Raspberry Pi. The models with the qualifying parameters were trained again. The second phase experiments were conducted five times for each parameter combination. Subsequently, the mean values of training time, training accuracy, and test accuracy of five repeated experiments were recorded.

During the experiments, the performances of the CPU and RAM are checked occasionally. All four cores of the CPU were involved during the training of models. In the following section, the structure of the networks and the tunning process of hyperparameters are explained.

4.1. Structure of the Networks

The general structure of the CNN's and ELM's were preserved during the experiments. To determine the number of nodes and batch size for the experiments, numbers that are the power of two are selected. Such selection for the parameters is optimal for the memory allocation and thus for models. Besides, the performance of processors is better with such selection [23]. Furthermore, the variety for searching space for the parameters increases drastically by selecting the parameters as a power of two.

4.1.1. Convolutional Neural Networks

The models with large size and computational complexity were avoided in order to provide a fair comparison of models on Raspberry Pi.

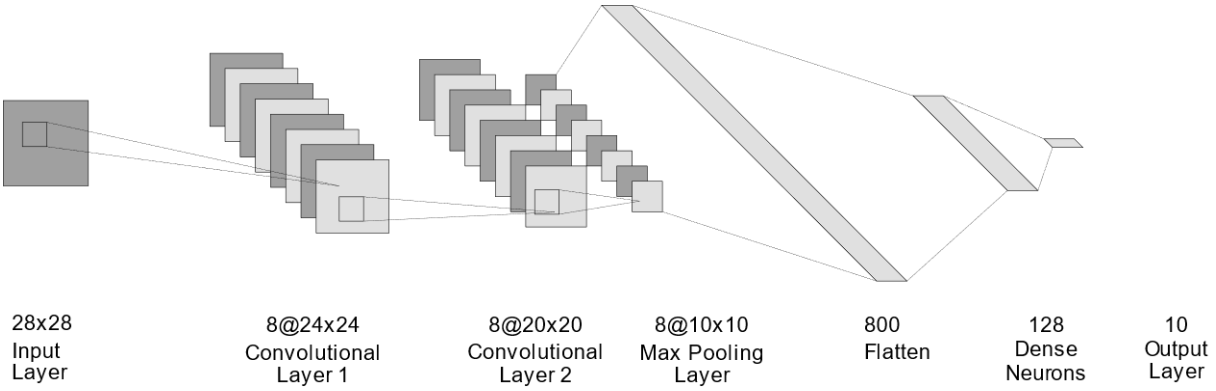

Figure 4.2 Convolutional Neural Network Diagram. The diagram represents a specific architecture used in thesis experiments.

For the experiments, as a base architecture, the above model in Figure 4.2 was used. The architecture consists of 6 layers: Input layer, two convolutional layers, max-pooling layer, hidden layer, and output layer. The input is based on grayscale images with the dimensions of 28x28, provided by the MNIST and Fashion MNIST datasets. Each image in both datasets consists of 784 pixels. Thus, both convolutional layers consist of 8 filters with a 5x5 size and a stride of 1. The fourth layer, the max-pooling layer, consists of a pooling size of 2 x 2 and stride of 2. Between the fully connected layer and max-pooling layer, the image is flattened. There is only one fully connected layer, which contains at base 128 neurons, followed by the ReLu function. In the end, there is an output layer with ten outputs and a Softmax function.

For the experiments, Adam algorithm [43] with a learning rate of 0.001 was used for the optimization, which is the default setting for the TensorFlow framework. The kernels were initialized with the Glorot initializer [81], and the biases were initialized with "zeros". There was no "dropout" added to the network. Moreover, no padding was used since the shrinkage caused by convolutional layers was tolerable. The batch size was 32. This means

[23] Intel. *CIFAR-10 Classification using Intel® Optimization for TensorFlow**. 2017, December 13; Available from: https://software.intel.com/.

that for a batch size of 32 and sample number of 50,000, the iteration number for each epoch is 1563. There were no parameters for the pooling layer to be tuned.

In the preliminary experiments, in the first stage, the following parameters were alternated:

- Number of epochs (2,4,16)
- Number of hidden nodes in the fully connected layer (128, 256, 512 and 1024)

The maximum number of epochs is restricted to 16 since the training times for 16 epochs were significantly more than four epochs, particularly with the experiments conducted on Raspberry Pi, and the accuracy difference between the models was comparatively small. Considering both training time and accuracy as a performance metric, the performance of models with 16 epochs was relatively worse than the model with four epochs. Therefore, the trainings with a higher number of epochs are not conducted since the purpose of preliminary experiments is to find models with superior performance for the second stage of the experiments.

There are 105,634 parameters, which can be trained for the network. As a comparison, AlexNet, winner of the ImageNet Large Scale Visual Recognition Challenge in 2012 [2], has 62,378,344 parameters.

4.1.2. Extreme Learning Machines

Since the Extreme Learning Machine algorithm is non-iterative, there were not many types of hyperparameters involved in training. Therefore, the only hyperparameter that needs to be alternated was the hidden node count for the layers. In the case of Single Layer Extreme Learning machines, there was only one parameter, and for the Two-Layer Extreme Learning Machines models, there were two parameters to be tuned. In Figure 4.3, the structure of the Single-Layer ELM, which was used in the experiments, can be seen.

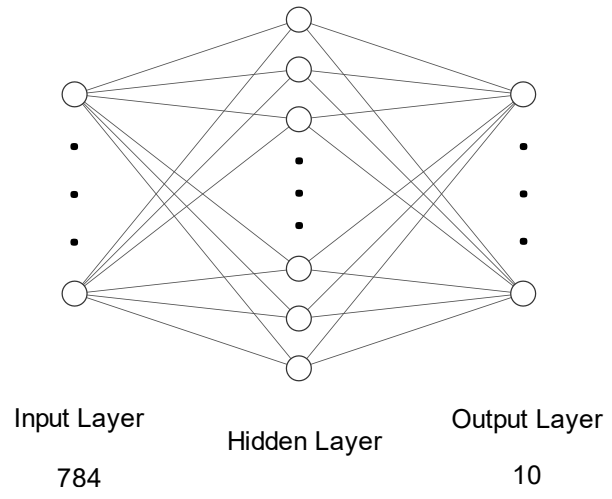

Figure 4.3 Single Hidden Layer Extreme Learning Machine Diagram

As an activation function for the hidden layer, the sigmoid function was used. The input images consist of 784 pixels; thus, the input layer consists of 784 input neurons. MNIST and Fashion MNIST datasets have ten output classes. Therefore, the output size of the ELM was ten. The weights and biases were initialized using modified He initialization [82] instead of Glorot initialization (also referred to as Xavier initialization) [81].

In the preliminary experiments, the following hidden neuron numbers were used for the Single Layer ELM model: 128, 256, 512, 1024, 2048, 4096, 8192, and 16384. The experiments were conducted with the same settings on Raspberry Pi and the computer. Nevertheless, the experiments with 16384 hidden neurons could not be conducted on Raspberry Pi due to memory issues.

In Figure 4.4 below, the structure of the TELM used for the experiments can be seen.

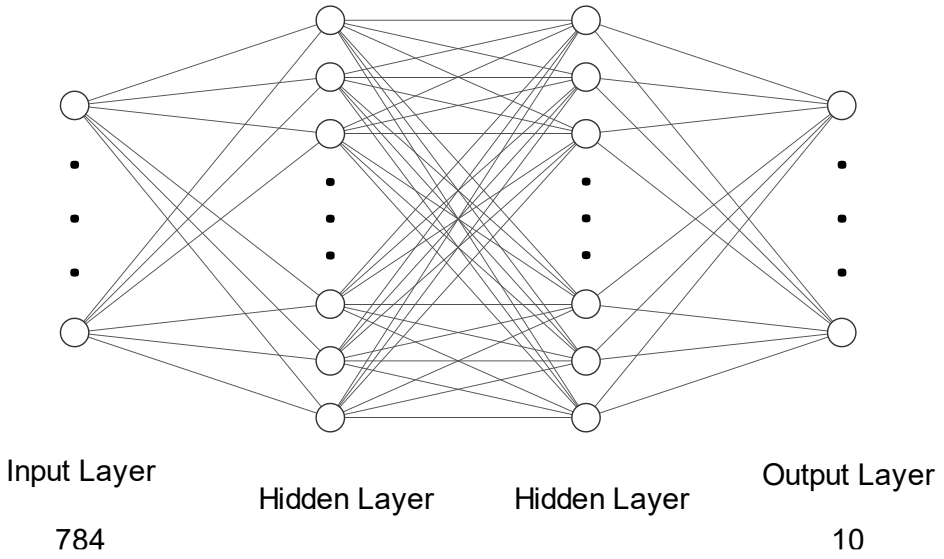

Figure 4.4 Two Hidden Layer Extreme Learning Machine Diagram

For the implementation of the TELM model, in the first hidden layer as an activation function, the sigmoid function was used, and in the second hidden layer, the ReLu function was used. The number of neurons in the input layer was 784, and the output size was 10. The weights and biases were initialized using modified He initialization [82].

In the preliminary experiments, the following hidden number combinations were used for the TELM model: 128-128, 256-256, 512-512, 1024-1024, 2048-2048, 4096-4096, and 8192-8192. The experiments were conducted with the same settings on Raspberry Pi and the computer.

In the second stage, experiments were conducted with the same settings. The number of hidden neurons was alternated, as explained at the beginning of the Methods section. The number of hidden neurons used can be seen in Figure 4.1 under “Tested parameter combinations”.

4.1.3. Hybrid Model

In the last phases of the research, the experiments were conducted on a model, which is a combination of CNN and ELM. Since the most time-expensive and computational power-demanding part of a CNN is backpropagation, the benefits of CNN (high accuracy) and ELM (fast computation and good generalization) are combined in this proposed method. The general structure of the process can be seen below in Figure 4.5.

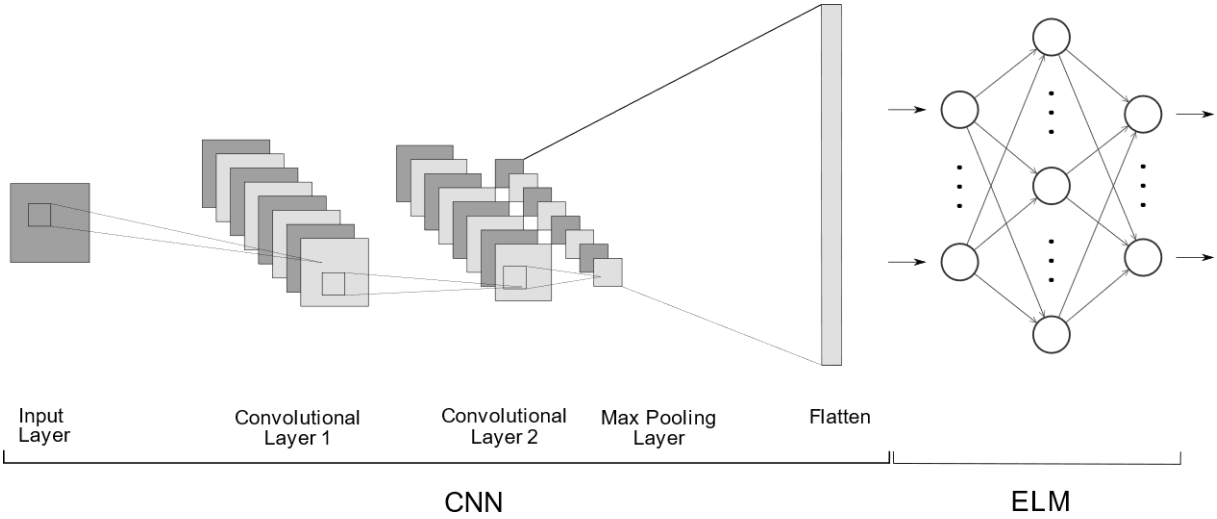

Figure 4.5 Combination of CNN and ELM Diagram. The diagram represents the general structure of the models, which are combination of CNN and ELM.

In order to give a fair comparison for the structure of the hybrid model, similar structures that are used for the previous experiment stages are used for CNN and ELM parts. The convolutional part of the hybrid model consists of two convolutional layers with a filter size of 5x5 and a filter count of 8 for each layer. For the second part of the hybrid model, Single Layer Extreme Learning Machine and Two-Layer Extreme Learning Machine were used independently with the same structure introduced in the previous sections.

In the experiments for the hybrid model consisting of CNN and Single-Hidden Layer Extreme Learning Machines, only the number of hidden neurons was adjusted. For the preliminary experiments on the computer, the following number of hidden neurons were investigated: 128, 256, 512, 1024, 2048, 4096, 8192, 16384. For the experiments with Raspberry Pi, same numbers were used. However, the experiments with 16384 hidden neurons could not be conducted due to memory problems, which could not be solved with performance improvement methods.

In the experiments for the hybrid model consisting of CNN and Two-Hidden Layer Extreme Learning Machines, the numbers of hidden neurons for the two hidden layers were adjusted. In the preliminary experiments for the computer, the following combinations of hidden neurons were investigated: 128-128, 256-256, 512-512, 1024-1024, 2048-2048, 4096-4096, 8192-8192. The same combinations were used for the experiments with Raspberry Pi. The

experiments with hidden neuron combinations of 8192-8192 and 16384-16384 could not be conducted due to memory problems.

In the second stage, experiments were conducted with the same settings. The number of hidden neurons used for the experiments can be seen in Figure 4.1 under “Tested parameter combinations”.

4.2. Metrics to compare the results

For the comparison of the models, the following metrics are used.

4.2.1. Training time

The training time is defined as the time required for the model's training, which consists of two stages: feature extraction and classification. For the hybrid model, training times are computed as the sum of the CNN and ELM parts. For the results, second is used as a unit.

4.2.2. Test Accuracy

The test accuracy is the classification accuracy of the model, which is calculated using test samples. It is the primary metric to evaluate the performance of models and can be calculated as:

$$Test\ Accuracy = \frac{Number\ of\ true\ predictions\ for\ test\ samples}{Total\ number\ of\ predictions\ for\ test\ samples} \quad (25)$$

4.2.3. Training Accuracy

Training accuracy is the classification accuracy of the model, which is calculated using the training samples. Since the model is tested on the same sample group used for the training, it is not used to evaluate the model's performance. Thus, training accuracies of the models are not included in the results; however, it is used as a secondary indicator of overfitting behavior. It can be calculated as:

$$Training\ Accuracy = \frac{Number\ of\ true\ predictions\ for\ training\ samples}{Total\ number\ of\ predictions\ for\ training\ samples} \quad (26)$$

4.2.4. Validation Accuracy

A validation set is used for the training of CNN models as the primary indicator of overfitting behaviour. The training data is split into training and validation. 80% of the training data is used for the training, and 20% of the training data is used for validation. For example, if the dataset consists of 60,000 training samples and 10,000 test samples, 12,000 samples are used for validation, and 48,000 samples are used for the training. The results for validation

accuracy are not included in the results. Validation accuracy can be determined using the equation:

$$Validation\ Accuracy = \frac{Number\ of\ true\ predictions\ for\ validation\ samples}{Total\ number\ of\ predictions\ for\ validation\ samples} \quad (27)$$

5. Results and Discussion

The results for each models with different datasets are grouped into one single table for each platform and shown in Table 1 to 3. Table 1, 2, and 3 shows the results for computer, Raspberry Pi, overclocked Raspberry Pi, respectively. As a performance metric, both training time and test accuracy were considered. It is of note to mention that lots of different architectures with different parameters were tested, and the results for relatively optimal architectures are reported in this section. The sections 5.1, 5.2, and 5.3 represent and discuss the results for different platforms. In section 5.4, an overview of the results and illustrative graphs are given based on selected data.

5.1. Results for the experiments on computer platform

Table 1. Results of the experiments on computer platform using two different data sets

Method	Subgroup	Test parameters	Dataset			
			MNIST		Fashion MNIST	
			TT (sec)*	TA (%)*	TT (sec)*	TA (%)*
CNN		Epoch/NN, 2/128	37.664	97.484	44.376	85.084
		Epoch/NN, 2/256	40.422	97.358	48.232	84.572
		Epoch/NN, 2/512	42.806	97.35	54.02	85.114
		Epoch/NN, 4/128	76.85	97.862	76.982	85.722
ELM	Single layer	NN, 2048	11.18	93.972	11.188	84.82
	Single layer	NN, 4096	55.506	94.454	63.364	86.174
	Single layer	NN, 8192	229.356	96.544	251.894	87.234
	Two layers	NN-NN, 2048-2048	40.232	94.888	44.434	83.44
	Two layers	NN-NN, 4096-4096	184.59	96.116	202.66	84.91
Hybrid	CNN plus single layer	NN, 2048	36.756	97.212	39.356	85.244
	CNN plus single layer	NN, 4096	80.488	97.58	94.516	86.108
	CNN plus single layer	NN, 8192	267.58	97.934	273.6	86.606
	CNN plus two layers	NN-NN, 1024-1024	40.296	95.354	42.544	82.076
	CNN plus two layers	NN-NN, 4096-4096	227.648	97.422	222.02	84.63

*Mean of five consecutive experiments. NN, number of neurons, CNN, convolutional neural networks; ELM, extreme learning machine; TT, test time; TA, test accuracy

The hybrid model with single-layer ELM and CNN model seems to perform comparable for the MNIST dataset. On the other hand, hybrid model with two-layer ELM performs poorly for both of the tasks. Most of the models performed better than human performance (83.5%) for the Fashion MNIST.

The performances of ELM models are worse than other methods for the MNIST dataset. Nevertheless, the results are opposite for the Fashion MNIST dataset.

Single Layer ELM outperforms other methods for the Fashion MNIST dataset on the computer platform. Furthermore, CNN models perform better than hybrid models for the Fashion MNIST dataset.

5.2. Results for the experiments on Raspberry Pi platform

Table 2. Results of the experiments on Raspberry Pi platform using two different data sets

Method	Subgroup	Test parameters	Dataset			
			MNIST		Fashion MNIST	
			TT (sec)*	TA (%)*	TT (sec)*	TA (%)*
CNN		Epoch/NN, 2/128	253.268	97.726	251.916	84.968
		Epoch/NN, 2/256	276.92	97.556	275.614	84.96
		Epoch/NN, 2/512	318.236	97.348	317.642	84.228
		Epoch/NN, 4/128	498.906	97.976	497.76	86.19
ELM	Single layer	NN, 2048	55.568	94.034	54.294	84.912
	Single layer	NN, 4096	204.292	95.524	199.066	86.18
	Single layer	NN, 8192	925.212	96.538	898.686	87.264
	Two layers	NN-NN, 2048-2048	186.48	94.906	182.85	83.398
	Two layers	NN-NN, 4096-4096	606.98	96.272	592.764	84.736
Hybrid	CNN plus single layer	NN, 2048	204.426	97.04	196.62	82.933
	CNN plus single layer	NN, 4096	353.428	97.592	347.396	84.866
	CNN plus single layer	NN, 8192	1075.668	98.068	1076.058	84.756
	CNN plus two layers	NN-NN, 1024-1024	225.26	95.952	216.138	81.878
	CNN plus two layers	NN-NN, 4096-4096	756.576	97.63	735.348	85.61

*Mean of five consecutive experiments. NN, number of neurons, CNN, convolutional neural networks; ELM, extreme learning machine; TT, test time; TA, test accuracy

The Single-Hidden Layer Extreme Learning Machine seems to perform better than the CNN algorithm for Fashion MNIST on Raspberry Pi as well. It can be concluded that as the input image gets complex, the ELM algorithm performs better for a specific configuration (NN,4096), since the performance of ELM is worse than CNN with the experiments for the MNIST dataset. On the other hand, the results of another study comparing the performance of ELMs and MLPs states that the performance of ELM degrades faster as the complexity of the dataset grows [83]. However, only accuracy was considered a performance parameter in that study, rather than also taking training time into consideration.

The Hybrid models perform better than ELM for the MNIST dataset generally. However, the performance of ELM models for the Fashion MNIST dataset is clearly better than Hybrid models.

5.3. Results for the experiments on Raspberry Pi platform with overclock

Table 3. Results of the experiments on Raspberry Pi platform using two different data sets with overclocking

Method	Subgroup	Test parameters	Dataset			
			MNIST		Fashion MNIST	
			TT (sec)*	TA (%)*	TT (sec)*	TA (%)*
CNN		Epoch/NN, 2/128	226.052	97.468	225.048	84.666
		Epoch/NN, 2/256	247.744	97.678	243.372	84.178
		Epoch/NN, 2/512	282.836	97.488	282.104	84.856
		Epoch/NN, 4/128	448.63	97.726	444.29	85.848
ELM	Single layer	NN, 2048	46.278	94.076	46.19	84.85
	Single layer	NN, 4096	168.204	95.512	168.05	86.258
	Single layer	NN, 8192	761.848	96.586	761.23	87.226
	Two layers	NN-NN, 2048-2048	154.788	94.852	155.114	83.292
	Two layers	NN-NN, 4096-4096	498.862	96.208	498.584	84.95
Hybrid	CNN plus single layer	NN, 2048	175.548	97.208	175.398	83.396
	CNN plus single layer	NN, 4096	297.778	97.5	298.118	84.32
	CNN plus single layer	NN, 8192	890.694	98.226	891.684	85.598
	CNN plus two layers	NN-NN, 1024-1024	193.87	95.834	193.676	82.79
	CNN plus two layers	NN-NN, 4096-4096	628.29	97.612	627.84	83.954

*Mean of five consecutive experiments. NN, number of neurons, CNN, convolutional neural networks; ELM, extreme learning machine; TT, test time; TA, test accuracy

The training times for Raspberry Pi is in average three times slower than the computer. Nevertheless, the results for the accuracy of computer and Raspberry Pi platforms are similar.

The overclocked Raspberry Pi experiments show that the training times are reduced (approximately 17% decrease of training time) with the increasing CPU clock speed. However, the accuracy performance remained the same.

5.4. Overview of the results

It seems that the hybrid models with two-layer ELM perform poorly on every experiment and does not provide desirable results. Single-layer ELM with 4096 neurons appears to be the best model performing for Fashion MNIST dataset on every platform. It should be considered that the experiments with single-layer ELM model for parameters more than 8192 could not be conducted on Raspberry Pi, due to memory issues. However, there were no such issues reported when conducting experiments with CNN. Another study [83] confirms that the ELMs require more hidden layer and larger network; thus, require more resources. The problem is mentioned in literature [84] and solutions are proposed [10].

The performance of two-layer ELM was worse than single-layer ELM on all experiments considering the training times, even though there are models which have higher accuracy.

However, the desired level of performance improvement with the hybrid model could not be reached in these tests. Similarly, another study [13] found that the unique ELM-CNN hybrid model performs worse than CNN with backpropagation as well, when applied on to large datasets such as MNIST.

In the following figures, Figure 5.1 and Figure 5.2, the best performing model configurations for each method, dataset, and platform are selected manually from the tables and illustrated separately for two datasets. The best performing models were same for different platforms.

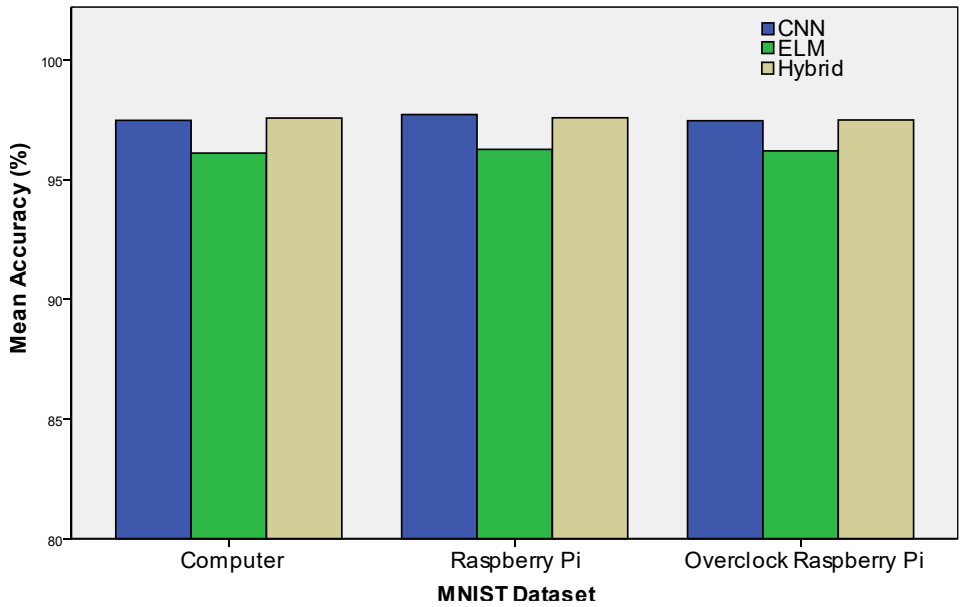

Figure 5.1 Accuracy performances of CNN, ELM, and hybrid methods on three different platforms using MNIST dataset. Bars represent the data for configurations selected as optimal considering accuracy as well as training times: CNN: Epoch 2, NN 128; ELM: Two layer, NN 4096/4096; Hybrid: CNN plus single layer, NN 4096. CNN and hybrid methods seem to perform similarly and better than ELM method.

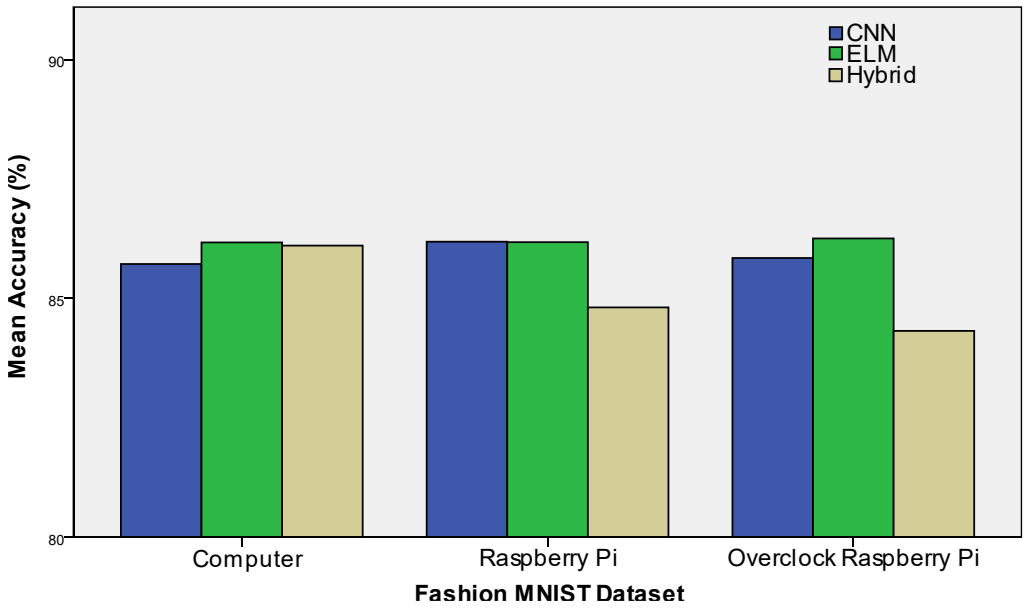

Figure 5.2 Accuracy performances of CNN, ELM, and hybrid methods on three different platforms using Fashion MNIST dataset. Bars represent the data for configurations selected as optimal considering accuracy as well as training times: CNN: Epoch 4, NN 128; ELM: single layer, NN 4096; Hybrid: CNN plus single layer, NN 4096. CNN and ELM methods seem to have better performances.

6. Conclusion

Findings of this study suggest that ELM performs better than CNN for Fashion MNIST dataset when both training time and accuracy are taken into account possibly owing to its simpler architecture. Considering that CNN models are constructed using highly optimized frameworks (by TensorFlow team), ELM has the potential to yield even better performance with similar optimizations.

In addition, all three methods, CNN, ELM, and a combination of them, appear to be suitable for training and classification tasks on mobile platforms. Potential application areas include IoT and embedded devices, particularly image classification tasks where training times as well as accuracy are important.

With increasing CPU clock speed, better performance is achieved on Raspberry Pi indicating that further performance improvement is possible with faster CPUs.

The performance test results for Single Layer ELM are promising so that it may represent a faster and accurate alternative to CNNs, particularly when relatively complex datasets are to be processed. However, these assumptions should be further tested on more complex datasets to be more conclusive.

7. Bibliography

1. Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein, *Imagenet large scale visual recognition challenge.* International journal of computer vision, 2015. **115**(3): p. 211-252.

2. Krizhevsky, A., I. Sutskever, and G.E. Hinton, *Imagenet classification with deep convolutional neural networks.* Advances in neural information processing systems, 2012. **25**: p. 1097-1105.

3. Hu, J., L. Shen, and G. Sun. *Squeeze-and-excitation networks*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

4. Zeng, Y., X. Xu, Y. Fang, and K. Zhao. *Traffic Sign Recognition Using Extreme Learning Classifier with Deep Convolutional Features*. 2015.

5. Huang, G., Q.-Y. Zhu, and C. Siew, *Extreme learning machine: a new learning scheme of feedforward neural networks.* 2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541), 2004. **2**: p. 985-990 vol.2.

6. Acquisti, A., L. Brandimarte, and G. Loewenstein, *Privacy and human behavior in the age of information.* Science, 2015. **347**(6221): p. 509-514.

7. Pan, S.J. and Q. Yang, *A Survey on Transfer Learning.* IEEE Transactions on Knowledge and Data Engineering, 2010. **22**(10): p. 1345-1359.

8. Kolesnikov, A., L. Beyer, X. Zhai, J. Puigcerver, J. Yung, S. Gelly, and N. Houlsby, *Big transfer (bit): General visual representation learning.* arXiv preprint arXiv:1912.11370, 2019. **6**(2): p. 8.

9. Ashton, K., *That 'internet of things' thing.* RFID journal, 2009. **22**(7): p. 97-114.

10. Yoo, Y. and S. Oh, *Fast training of convolutional neural network classifiers through extreme learning machines.* 2016 International Joint Conference on Neural Networks (IJCNN), 2016: p. 1702-1708.

11. Khellal, A., H. Ma, and Q. Fei, *Convolutional Neural Network Features Comparison Between Back-Propagation and Extreme Learning Machine.* 2018 37th Chinese Control Conference (CCC), 2018: p. 9629-9634.

12. Kim, J., J. Kim, G.-J. Jang, and M. Lee, *Fast learning method for convolutional neural networks using extreme learning machine and its application to lane detection.* Neural Networks, 2017. **87**: p. 109-121.

13. Khellal, A., H. Ma, and Q. Fei, *Convolutional Neural Network Based on Extreme Learning Machine for Maritime Ships Recognition in Infrared Images.* Sensors, 2018. **18**(5): p. 1490.

14. Gürpinar, F., H. Kaya, H. Dibeklioglu, and A.A. Salah. *Kernel ELM and CNN Based Facial Age Estimation*. in *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2016.

15. Yu, J.-s., J. Chen, Z. Xiang, and Y.-X. Zou. *A hybrid convolutional neural networks with extreme learning machine for WCE image classification*. in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2015. IEEE.

16. Jain, M., W. Andreopoulos, and M. Stamp, *Convolutional neural networks and extreme learning machines for malware classification.* Journal of Computer Virology and Hacking Techniques, 2020. **16**.

17. McCulloch, W.S. and W. Pitts, *A logical calculus of the ideas immanent in nervous activity.* The bulletin of mathematical biophysics, 1943. **5**(4): p. 115-133.

18. Ivakhnenko, A.G.e. and V.G.e. Lapa, *Cybernetics and forecasting techniques.* 1967.

19. Rumelhart, D.E., G.E. Hinton, and R.J. Williams, *Learning representations by back-propagating errors.* Nature, 1986. **323**(6088): p. 533-536.

20. Dechter, R., *Learning While Searching in Constraint-Satisfaction-Problems*. 1986. 178-185.

21. Hinton, G.E., S. Osindero, and Y.-W. Teh, *A fast learning algorithm for deep belief nets.* Neural computation, 2006. **18**(7): p. 1527-1554.

22. Wang, H. and B. Raj, *On the origin of deep learning.* arXiv preprint arXiv:1702.07800, 2017.

23. Wani, M.A., F.A. Bhat, S. Afzal, and A.I. Khan, *Advances in deep learning*. 2020: Springer.

24. Hubel, D.H. and T.N. Wiesel, *Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.* The Journal of physiology, 1962. **160**(1): p. 106-154.

25. Kamnitsas, K., C. Ledig, V.F.J. Newcombe, J.P. Simpson, A.D. Kane, D.K. Menon, D. Rueckert, and B. Glocker, *Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation.* Medical Image Analysis, 2017. **36**: p. 61-78.

26. Fukushima, K., *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.* Biological Cybernetics, 1980. **36**(4): p. 193-202.

27. Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 1998. **86**(11): p. 2278-2324.

28. LeCun, Y., *The MNIST database of handwritten digits.* http://yann. lecun. com/exdb/mnist/.

29. Simonyan, K. and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition.* arXiv 1409.1556, 2014.

30. Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.

31. He, K., X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

32. Huang, G., Z. Liu, L. Van Der Maaten, and K.Q. Weinberger. *Densely connected convolutional networks.* in *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

33. Xiao, H., K. Rasul, and R. Vollgraf, *Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.* arXiv preprint arXiv:1708.07747, 2017.

34. Krizhevsky, A. and G. Hinton, *Learning multiple layers of features from tiny images.* 2009.

35. Misra, D., *Mish: A self regularized non-monotonic neural activation function.* arXiv preprint arXiv:1908.08681, 2019. **4**.

36. Tang, Y., *Deep learning using linear support vector machines.* arXiv preprint arXiv:1306.0239, 2013.

37. Linnainmaa, S., *Taylor expansion of the accumulated rounding error.* BIT Numerical Mathematics, 1976. **16**(2): p. 146-160.

38. Werbos, P.J., *Applications of advances in nonlinear sensitivity analysis*, in *System modeling and optimization*. 1982, Springer. p. 762-770.

39. Robbins, H. and S. Monro, *A Stochastic Approximation Method.* The Annals of Mathematical Statistics, 1951. **22**(3): p. 400-407.

40. Ruder, S., *An overview of gradient descent optimization algorithms.* arXiv preprint arXiv:1609.04747, 2016.

41. Kiefer, J. and J. Wolfowitz, *Stochastic Estimation of the Maximum of a Regression Function.* The Annals of Mathematical Statistics, 1952. **23**(3): p. 462-466.

42. Duchi, J., E. Hazan, and Y. Singer, *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.* J. Mach. Learn. Res., 2011. **12**(null): p. 2121–2159.

43. Kingma, D.P. and J. Ba, *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980, 2014.

44. Tieleman, T. and G. Hinton, *Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.* COURSERA: Neural networks for machine learning, 2012. **4**(2): p. 26-31.

45. LeCun, Y., Y. Bengio, and G. Hinton, *Deep learning.* Nature, 2015. **521**(7553): p. 436-444.

46. Huang, G.-B., Q.-Y. Zhu, and C.-K. Siew, *Extreme learning machine: Theory and applications.* Neurocomputing, 2006. **70**(1): p. 489-501.

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

48. Huang, G.-B., *An insight into extreme learning machines: random neurons, random features and kernels.* Cognitive Computation, 2014. **6**(3): p. 376-390.

49. Cao, J., Z. Lin, G.-B. Huang, and N. Liu, *Voting based extreme learning machine.* Information Sciences, 2012. **185**(1): p. 66-77.

50. Huang, G.-B., N. Liang, H.-J. Rong, P. Saratchandran, and N. Sundararajan, *On-Line Sequential Extreme Learning Machine*. Vol. 2005. 2005. 232-237.

51. Zhu, Q.-Y., A.K. Qin, P.N. Suganthan, and G.-B. Huang, *Evolutionary extreme learning machine.* Pattern Recognition, 2005. **38**(10): p. 1759-1763.

52. Pao, Y.-H., G.-H. Park, and D.J. Sobajic, *Learning and generalization characteristics of the random vector functional-link net.* Neurocomputing, 1994. **6**(2): p. 163-180.

53. Schmidt, W.F., M.A. Kraaijveld, and R.P. Duin. *Feed forward neural networks with random weights*. in *International Conference on Pattern Recognition*. 1992. IEEE COMPUTER SOCIETY PRESS.

54. Yam, J.Y.F. and T.W.S. Chow, *A weight initialization method for improving training speed in feedforward neural network.* Neurocomputing, 2000. **30**(1): p. 219-232.

55. Huang, G.-B., *What are Extreme Learning Machines? Filling the Gap Between Frank Rosenblatt's Dream and John von Neumann's Puzzle.* Cognitive Computation, 2015. **7**: p. 263-278.

56. Qu, B.Y., B.F. Lang, J.J. Liang, A.K. Qin, and O.D. Crisalle, *Two-hidden-layer extreme learning machine for regression and classification.* Neurocomputing, 2016. **175**: p. 826-834.

57. Huang, G., *Learning capability and storage capacity of two-hidden-layer feedforward networks.* IEEE transactions on neural networks, 2003. **14 2**: p. 274-81.

58. Pajankar, A., *Raspberry Pi Supercomputing and Scientific Programming: MPI4PY, NumPy, and SciPy for Enthusiasts*. 2017: Apress.

59. Monk, S., *Electronics Cookbook: Practical Electronic Recipes with Arduino and Raspberry Pi*. 2017: " O'Reilly Media, Inc.".

60. Maksimovic, M., V. Vujovic, N. Davidović, V. Milosevic, and B. Perisic, *Raspberry Pi as Internet of Things hardware: Performances and Constraints*. 2014.

61. Atzori, L., A. Iera, and G. Morabito, *The Internet of Things: A survey.* Computer Networks, 2010. **54**(15): p. 2787-2805.

62. Kochláň, M., M. Hodoň, L. Čechovič, J. Kapitulík, and M. Jurečka. *WSN for traffic monitoring using Raspberry Pi board*. in *2014 Federated Conference on Computer Science and Information Systems*. 2014.

63. Rajsuman, R., *System-on-a-chip: Design and Test*. 2000: Artech House, Inc.

64. Ryzhyk, L., *The ARM Architecture.* Chicago University, Illinois, EUA, 2006.

65. Upton, E. and G. Halfacree, *Raspberry Pi user guide*. 2014: John Wiley & Sons.

66. Liu, S., Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen. *Cambricon: An Instruction Set Architecture for Neural Networks*. in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 2016.

67. Schlegel, D., *Deep Machine Learning on GPU*. 2015, University of Heidelberg, ZITI.

68. Mittal, S., *A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform.* Journal of Systems Architecture, 2019. **97**: p. 428-442.

69. Halfacree, G., *Thermal Testing Raspberry Pi 4*, in *The MagPi Issue 88*. 2019, Raspberry Pi Press. p. 66-74.

70. Srinath, K., *Python–the fastest growing programming language.* International Research Journal of Engineering and Technology, 2017. **4**(12): p. 354-357.

71. Walt, S.v.d., S.C. Colbert, and G. Varoquaux, *The NumPy Array: A Structure for Efficient Numerical Computation.* Computing in Science & Engineering, 2011. **13**(2): p. 22-30.

72. Virtanen, P., R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, and J. Bright, *SciPy 1.0: fundamental algorithms for scientific computing in Python.* Nature methods, 2020. **17**(3): p. 261-272.

73. Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, and V. Dubourg, *Scikit-learn: Machine learning in Python.* the Journal of machine Learning research, 2011. **12**: p. 2825-2830.

74. Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard. *Tensorflow: A system for large-scale machine learning*. in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016.

75. François, C., *Keras: The Python deep learning library.* keras. io, 2015.

76. Howard, J. and S. Gugger, *Fastai: A Layered API for Deep Learning.* Information, 2020. **11**(2): p. 108.

77. LeCun, Y., L.D. Jackel, L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, U.A. Muller, E. Sackinger, and P. Simard, *Learning algorithms for classification: A comparison on handwritten digit recognition.* Neural networks: the statistical mechanics perspective, 1995. **261**(276): p. 2.

78. Byerly, A., T. Kalganova, and I. Dear, *A branching and merging convolutional network with homogeneous filter capsules.* arXiv preprint arXiv:2001.09136, 2020.

79. Tanveer, M.S., M.U.K. Khan, and C.-M. Kyung, *Fine-Tuning DARTS for Image Classification.* arXiv preprint arXiv:2006.09042, 2020.

80. Ylonen, T. and C. Lonvick, *The secure shell (SSH) protocol architecture*. 2006, RFC 4251, January.

81. Glorot, X. and Y. Bengio. *Understanding the difficulty of training deep feedforward neural networks*. in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010. JMLR Workshop and Conference Proceedings.

82. He, K., X. Zhang, S. Ren, and J. Sun. *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*. in *Proceedings of the IEEE international conference on computer vision*. 2015.

83. Castellani, A., S. Cornell, L. Falaschetti, and C. Turchetti. *tfelm: a TensorFlow Toolbox for the Investigation of ELMs and MLPs Performance*. in *Proceedings on the International Conference on Artificial Intelligence (ICAI)*. 2018. The Steering Committee of The World Congress in Computer Science, Computer ….

84. Chen, C., K. Li, M. Duan, and K. Li, *Chapter 6 - Extreme Learning Machine and Its Applications in Big Data Processing*, in *Big Data Analytics for Sensor-Network Collected Intelligence*, H.-H. Hsu, C.-Y. Chang, and C.-H. Hsu, Editors. 2017, Academic Press. p. 117-150.