ADAPTIVE ACTIVATIONS AND SHIFT INVARIANCE IN SHALLOW CONVOLUTIONAL NEURAL NETWORKS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF UNIVERSITY OF TEXAS AT ARLINGTON
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Chinmay Appa Rane
August 2021

To my Father, my Mother
my Brother, my Niece and my Wife

Abstract

Deep learning training training algorithms are a huge success in recent years in many fields including speech, text,image video etc. Deeper and deeper layers are proposed with huge success with resnet structures having around 152 layers. Shallow convolution neural networks(CNN's) are still an active research, where some phenomena are still unexplanined. CNN's are assumed to be invariant to shift due to its architecture, but recent studies have shown other wise. Apart from shift invariance, activation functions used in the network are of utmost importance, as they provide non linearity to the networks. Relu's are the most commonly used activation function.

We show a shallow network which is specifically used for classifying images with shifted objects. Completed Tasks are shown for analyzing and improving shallow networks shift invariance in convolutional neural networks. We demonstrate commonly used downsampling technique and show if these downsampling techniques work for shallow CNN's. We also show a way to factorize the output weights in the feature layer. A traditional segmentation example is shown for the shifted objects and subsequent results are also given.

We also show a complex piece-wise linear(PWL) activation in the hidden layer. We show that these PWL activations work much better than relu activations in our networks for convolution neural networks and multilayer perceptrons. Result comparison in MATALB and PYTORCH for shallow and deep CNNs are given to further strengthen our case. A naive growing and pruning algorithm for these PWL activation is shown and compared with original results.

Acknowledgments

I would like to express my sincere gratitude to Dr. Michael T Manry for his continuous support and guidance throughout my Master's thesis and Doctoral research. I am grateful to him for giving me the opportunity to work in the IPNNL lab, and providing research and financial support, which helped me throughout my research. I would also like to thank Dr. Manry for teaching me how to solve complex problems by cutting them into smaller pieces to assemble them into simple concepts. I admire his patience and insistence on perfection, which helped me throughout my journey.

I would like to also sincerly thank Dr. Venkat Deverajan, Dr. Ioannis D Schizas, Dr. Ramtin Madani and Dr. Daniel S Levine for taking time to serve on my dissertation committee. I would like to personally thank Miss Gail Panuski for helping me through my GTA assistantship and all related help throughout my time in academia.

I would like to thank my wife Stacy Marshall, without whom I wouldn't have been able to accomplish my dreams. I would like to thank her for listening to all my problems and sticking with me throughout my journey even during difficult times. I specially like to thank my mom, my dad, and my brother for financially and mentally supporting me and believing in me throughout my life's journey. Special thanks to my sister-in-law and my niece for those continuous Snapchat and Whatsapp video calls. It always helped me calm my nerves.

A very special thanks to Dr Kanishka Tyagi for guiding me through my research and papers in my final days of my dissertation. I would like to also thank my colleagues Rohit Rawat, Son Nguyen, and Yash Shinge for the help and support during my master and PhD years.

Finally, I would like to thank my friends Sudhir Menon, Nitin Sankapanavar, Akshay Tyagi, Soham Umbhrajkar and special thanks to Jugal Sheth for taking out time from his busy schedule to help with concepts. Also, thanks to all my childhood friends Sushil Chavan, Swara Jamdar, Ushmi Shah, Ritika Salian for all the late night calls. I would like to also thank anyone whom I might have missed.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, deep learning architectures have found tremendous success in speech recognition, image recognition, language recognition and translation. For speech, languague recognition and translation, deep recurrent neural networks(RNNs)[1][2][3] have shown improvements over older technologies. These networks can not only process images but also can process sequences of images such as videos, text and speech. The RNN structure consists of cells and gates. These cells store important information over time and the gates decide the passage of information in and out of the cells. RNNs are faced with vanishing gradient problems and cannot process words over a long period of time. To address this situation, networks such as long shot term memory(LSTM)[4], gated recurrent unit(GRU)[5] and transformers[6] are shown. These networks are designed to handle long sequences over time and are also used for image processing. Convolutional Neural networks(CNNs)([7], [8], [9]) are more widely used in image based applications. Combination networks such as CNN-LSTM, CNN-transfomer are also used for visual recognition[10][11] and time series analysis[12]

CNNs are used for image based application as a feature extractor, where one doesnt need to explicitly extract features for classifying images. Applications for CNNs include in diabetic retinopathy screening[13], lesion detection[14][15], skin lesion classification[16][17], human action recognition[18] [19], face recognition [20] [21], document analysis[22] [23] and in many other applications. CNNs can be trained using gradient approaches such as back propagation ([24], [25], [26], [27]), and conjugate gradient ([28],[29],[30], [31]).

Despite their popularity, CNNs still have some limitations such as its poorly understood shift - invariance, overfitting of the data, and the use of oversimplified nonlinear activation functions such as relu [32],and leaky relu [33][34].

Shift invariance in CNNs is still an active research area, since shift by one pixel in an object can dramatically change the output discriminants of the network [35]. This is due to aliasing. Although the convolution operator is shift invariant, the classification layer typically lack shift invariant property. Most commonly used downsampling methods such as pooling[36][37] partially solve the problems but cannot be used in shallow networks as they do not capture high level features.

CNNs face problems with selection of the number of filters, filter size, choice of activation function, the depth of the convolutional layers, use of batch normalization, regularization, downsampling methods, number of fully connected MLP layer's hidden units. One cannot simply use batch normalization, regularization and down sampling methods in all layers. Many modern architectures implement their own structures to use these techniques. Hyperparameter selection is still a big problem faced by CNNs.

Nonlinear activation functions such as relu[32] and leaky relu [33] have been widely used in a number of computer vision[38] and deep neural network[39] applications. These activation functions are not as complex as sigmoids [40] or hyperbolic tangent functions(tanh)[41] but are favored because they partially solve the vanishing gradient problem [42]. These relu activations do not guarantee optimal results as different sets of activations can lead to optimal results for each of the filters. For example. A CNN for a image classication application with 20 filters might need 20 different activations. The number of filters required for a particular application is not known.

Although, these activations lead to universal approximation[43] in mutlilayer perceptrons, many attempts have been made to create adaptive or fixed piecewise linear activation function [[44], [45], [46] ,[47] ,[48]]. Adaptive activation functions for deep CNNs are introduced in[49], where the author trains the slope and hinges on the curve using gradient descent techniques. The author has shown promising results in terms of testing accuracies in CIFAR-10 and CIFAR-100 image recognition dataset[50] and high-energy physics involving Higgs boson decay modes[51].

In this dissertation, we investigate methods for improving shift invariance (SI) in CNN's and also investigate a trainable piecewise linear (PWL) activation, since they can approximate the optimal mix of activations through universal approximation. In section II, we briefly review the multilayer perceptron's (MLP's) architecture, notation, training and its properties. In section III, we review the CNN architecture and notation as well as its back propagation algorithm. In Section IV, we explain two problems with CNN's and give goals associated to the problems. Section V, we elaborate progress made on Goal 1 Section VI, explains progress made on Goal 2. Finally, In Section VII, we discuss additional work and conclude this dissertation.

Chapter 2

Review: Multilayer Perceptron With Single Hidden Layer

In this chapter, we review our notation for a single hidden layer cascade connected MLP, briefly summarize several commonly used feedforward classifier training methods and describe some of the MLP's properties.

2.1 MLP notation and structure

A cascade connected MLP with one hidden layer is shown in figure 2.1. Input weight $w(k, n)$ connects the n^{th} input to the k^{th} hidden unit. Output weight $w_{oh}(i, k)$ connects the k^{th} hidden unit's activation $o_p(k)$ to the i^{th} output. $y_p(i)$ is the p^{th} pattern, i^{th} output activation which is linear activation in figure2.1. In the training pattern $\{\mathbf{x}_p, \mathbf{t}_p\}$ for a MLP, the p^{th} input vector $\mathbf{x}_p$ is initially of dimension N and the p^{th} desired output (target) vector $\mathbf{t}_p$ has dimension M. The pattern number p varies from 1 to N_v. The threshold is handled by augmenting $\mathbf{x}_p$ with an extra element $X_p(n+1)$ which is equal to one where, $\mathbf{x}_p = [x_p(1), x_p(2), ..., x_p(N+1)]^T$

For the p^{th} pattern, the k^{th} hidden unit's net function $n_p(k)$ is then

$$n_p(k) = \sum_{n=1}^{N+1} w(k, n) \cdot x_p(n) \tag{2.1}$$

which can be summarized as

$$\mathbf{n}_p = \mathbf{W} \cdot \mathbf{x}_p \tag{2.2}$$

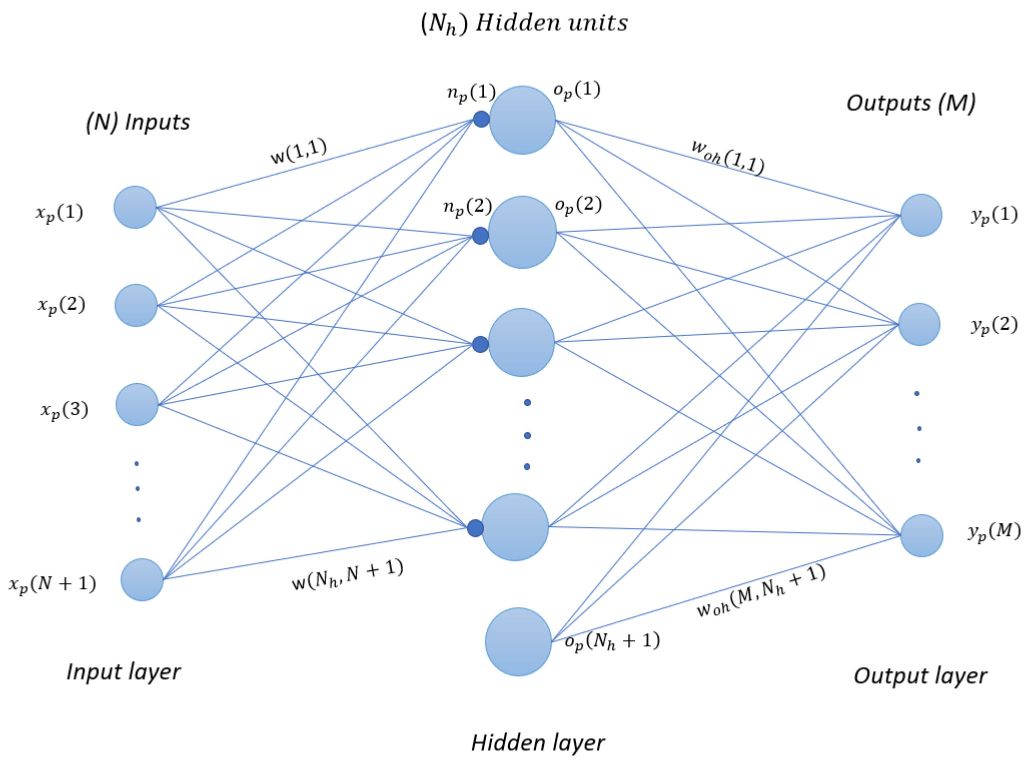


Figure 2.1: Single Hidden Layer MLP

where $\mathbf{n}_p$ denotes the N_h dimensional column vector of net function values and the input weight matrix $\mathbf{W}$ is N_h by (N+1). For the p^{th} pattern, the k^{th} hidden unit's output, $o_p(k)$, is given as

$$o_p(k) = f(n_p(k)) \tag{2.3}$$

where $f(.)$ denotes a nonlinear hidden layer activation function, such as relu[32] which is represented as

$$f(n_p(k)) = \begin{cases} n_p(k), & if \quad n_p(k) \geq 0 \\ 0, & if \quad n_p(k) < 0 \end{cases} \tag{2.4}$$

The threshold in the hidden layer is handled by augmenting $\mathbf{o}_p$ with an extra element $o_p(N_h+1)$ which is equal to one where $\mathbf{o}_p = [o_p(1), o_p(2), ..., o_p(N_h+1)]^T$. The network's output vector for the p^{th} pattern is $\mathbf{n}_{po}$. The i^{th} element $n_{po}(i)$ of the M-dimensional output vector $\mathbf{n}_{po}$ is

$$n_{po}(i) = \sum_{k=1}^{N_h+1} w_o(i,k) \cdot o_p(k) \tag{2.5}$$

which can be summarized as

$$\mathbf{N}_{po} = \mathbf{W}_o \cdot \mathbf{o}_p \tag{2.6}$$

$\mathbf{W_o}$ is the output weight matrix with dimensions M by $(N_h + 1)$.

The output layer net vector $\mathbf{n}_{po}$ is passed through an activation function for i^{th} output which is given in terms of p^{th} pattern and i^{th} output, $y_p(i)$ as,

$$y_p(i) = f_o(\mathbf{n}_{po}(i)) \tag{2.7}$$

where $f_o(.)$ denotes a output hidden layer activation function. The most commonly used activation function for approximation data is the linear activation, sigmoid activation is mostly used in logistic regression and softmax activation [52] is used for classification models which is defined in equation (2.8).

The softmax output activation function is

$$y_p(i) = \frac{exp(n_{po}(i))}{\sum_{k=1}^{M} exp(n_{po}(k))} \tag{2.8}$$

The most commonly used objective function for a classification task is cross entropy loss function[52]

$$E_{ce} = \frac{1}{N_v}\sum_{p=1}^{N_v}[-\sum_{i=1}^{M} t_p(i) \cdot log(y_p(i))] \tag{2.9}$$

where $t_p(i)$ is the p^{th} pattern and i^{th} class one hot encoded output. $t_p(i)$ is found from $ic_p(i)$, where $ic_p(i)$ is the class number. For approximation or regression, mean square error(MSE)[53] defined in equation (2.10) is the most widely used objective function. The MSE can also be used in the classification task with output reset[54][55].

$$E = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - y_p(i)]^2 \tag{2.10}$$

2.2 Gradient Training Approaches

In this section, we briefly summarize the steepest descent and conjugate gradient training methods

2.2.1 Steepest Descent

Steepest descent (SD) is the simplest optimization technique. SD was first suggested by Cauchy in 1847 [56], but its convergence properties for non-linear optimization problems were first studied by Haskell Curry in 1944[57]. It is an iterative process that attempts to find a local minimum of a objective function by taking steps proportional to the gradient of the function. To train an MLP using SD ,the network weight vector $\mathbf{w}$ is updated as,

$$\mathbf{w} \leftarrow \mathbf{w} + z \cdot \mathbf{g} \tag{2.11}$$

where, $\mathbf{w} = vec(\mathbf{W}, \mathbf{W_o})$, where $\mathbf{W}$ and $\mathbf{W_o}$ are input and output weight matrices. $\mathbf{g}$ is the negative gradient defined as

$$\mathbf{g} = -\frac{\partial E}{\partial \mathbf{w}} \tag{2.12}$$

where, E is the objective function defined in equation (2.10), and where (2.9). z is the learning factor. An optimal learning factor can be determined using the Gauss-Newton method as [58]

$$z = -\frac{\frac{\partial E(\mathbf{w}+z\cdot\mathbf{g})}{\partial z}}{\frac{\partial^2 E(\mathbf{w}+z\cdot\mathbf{g})}{\partial z^2}} \tag{2.13}$$

2.2.2 Conjugate Gradient Training

Conjugate gradient (CG) is a well-known unconstrained optimization technique, and its use in efficiently training an MLP is well documented [[30], [31],[40],[59], [60]]. The CG algorithm performs line-searches in the successive conjugate directions and has faster convergence than SD. To train an MLP using conjugate gradient, the network weights $\mathbf{w}$ are updated as.

$$\mathbf{w} \leftarrow \mathbf{w} + z \cdot \mathbf{p} \tag{2.14}$$

where, the direction vector $\mathbf{p}$ is obtained from the negative gradient $\mathbf{g}$ as

$$\mathbf{p} \leftarrow \mathbf{g} + B_1 \cdot \mathbf{p} \tag{2.15}$$

where $\mathbf{p} = vec(\mathbf{P}, \mathbf{P_o})$ and where $\mathbf{P}$ and $\mathbf{P_o}$ are the direction matrices corresponding to the weight arrays $\mathbf{W}$ and $\mathbf{W_o}$. B_1 is the ratio of the gradient energy from two consecutive iterations. z is an optimal learning factor[58] and can be derived from (2.16) and (2.14) as,

$$z = -\frac{\frac{\partial E(\mathbf{w}+z\cdot\mathbf{p})}{\partial z}}{\frac{\partial^2 E(\mathbf{w}+z\cdot\mathbf{p})}{\partial z^2}} \tag{2.16}$$

If the objective function is quadratic, CG converges in N_w iterations [30] [60], where the number of network weights satisfies $N_w = dim(\mathbf{w})$.

2.3 Multilayer Perceptron Theory

There are two classical theorem that summarizes the capabilities of MLPs

Theorem 1 : An MLP with a single hidden layer with any squashing activation function can approximate any continuous function uniformly on a compact set arbitary well.

Although the MLP structure has good approximate properties there is no guarantee that the user can find a training method that generates the desired approximation [43] [61][40].

Let $d_i(\mathbf{x})$ denote Type B_3 Bayes optimal discriminant, $P(i|\mathbf{x})$, the probability that the given vector $\mathbf{x}$ belongs to the i^{th} class[62], where $\mathbf{x}$ denote inout vector. N_c denotes the number of output classes. Let $y_i(\mathbf{x}_p)$ denote the i^{th} MLP output for the p^{th} input vector $\mathbf{x}_p$. The average squared error for MLP classifier training is given as

$$E(\mathbf{w}) = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{N_c} [t_p(i) - y_i(\mathbf{x}_p)]^2 \tag{2.17}$$

Similarly, the expected squared error between network outputs and Bayes discriminants is

$$e(\mathbf{w}) = \sum_{i=1}^{N_c} E[(y_i(\mathbf{x}) - d_i(\mathbf{x}))^2] \tag{2.18}$$

Theorem 2 : As the number appropriate training patterns N_v increases, the training error $E(\mathbf{w})$ approaches $e(\mathbf{w}) + C$, where C is a constant. Unfortunately, Theorem 2 does not require that $y_i(x)$ is an MLP output. It also does not specify N_h, and does not specify a training algorithm. It implies that the result applies to any technique which attempts to minimize MSE with desired outputs as one and zero for the multiclass problem[62][63].

2.4 Problems with MLP Classifiers

MLP's are widely used for many image based applications. Objective functions such as softmax cross entropy objective functions have found increasing success instead of MSE objective functions due to its output probability. Below we discuss some of the limitations of MLPs for image classification.

- When conjugate gradient training is used, the number of networks weights(N_w) increases, the number of training iterations require is N_w, which is quadratic in nature.[64].
- Suppose that a fully connected MLP's inputs are image pixels. As N_h and number of hidden layers N_L increases linearly the newtork pattern storage increases exponentially. For example, if an input image is of size 28 by 28 by 3, each of the pixels are connected to each of the hidden units in the next layer, so here the pattern storage of the weights is 28 by 28 by 3.
- Recognition in the previous problem may not require that every hidden unit be connected to every pixel. Hence N_w may need to be reduced so as to avoid overtraining.
- Similarly, if the adjacent pixel association is important during training, MLPs can perform poorly as MLPs do not extract features based on nearby pixels.
- MLPs for image classification lack shift invariance. This means that if an image is shifted from its original position, the output discriminants change.

Chapter 3

Review: Convolutional Neural Networks(CNN)

In this chapter, we first review notation and training of a convolutional neural network with a single convolution layer. Then we extend the notation to cover CNNs with multiple hidden layers.

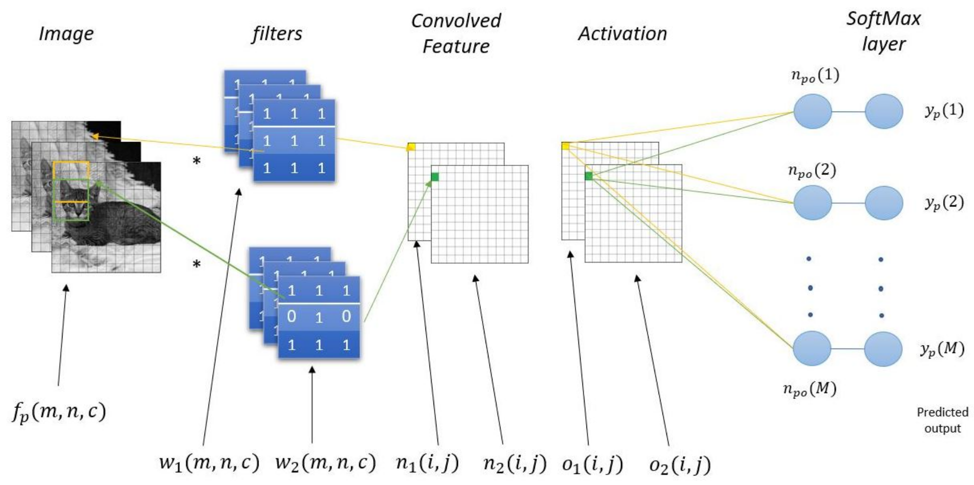


Figure 3.1: Shallow CNN with Linear softmax cross-entropy classifier

3.1 CNN structure and Notation

The CNN network structure is shown in figure 3.1. Let $\mathbf{f}_p$ denote the p^{th} input image and let $i_c(p)$ denote the correct class number of the p^{th} pattern, where p varies from 1 to N_v, and N_v is the total number of training images or patterns.

During forward propagation, a filter of size N_f x N_f is convolved over the image f_1 with N_r rows, N_c columns.The number of channels is denoted by C, where color input images have C equal to 3 and grayscale images have C equal to 1.

For the k^{th} filter, the net function output for the i^{th} row and j^{th} column is

$$n_p(k,i,j) = t_r(k) + \sum_{m=1}^{N_f}\sum_{n=1}^{N_f}\sum_{c=1}^{C} w_f(k,m,n,c)\cdot f_p(m+(i-1)s, n+(j-1)s, c) \tag{3.1}$$

where, $\mathbf{n}_p$ is of size (K by M_o by N_o), where K is the number of filters, M_o is the height of the convolved image output and N_o is the width of the convolved image output. $w_f(k,m,n,c)$ is the filter of size (K by N_f by N_f by C). The threshold vector $\mathbf{t_r}$ is added to the net function output as shown in equation (3.1). The stride s is the number of filter shifts over input images. Note that the output $n_p(k,i,j)$ in 3.1 us a threshold plus a sum of C separate 2-D convolution, rather than a 3-D convolution.

To achieve non-linearity, the convolved image with element $n_p(k,i,j)$ is passed through a relu activation [32] as

$$o_p(k,i,j) = f'(n_p(k,i,j)) \tag{3.2}$$

where, $o_p(k,i,j)$ is the k^{th} filter's hidden unit activation output for the i^{th} row, j^{th} column for the p^{th} pattern of size (K by N_{rb} by N_{cb}), where N_{rb} by N_{cb} is the row and column size of the output of the convolved image respectively.

The net function $\mathbf{n}_{po}$ for i^{th} element of the CNN's output layer for the p^{th} pattern is

$$n_{po}(i) = t_o(i) + \sum_{m=1}^{M_o}\sum_{n=1}^{N_o}\sum_{k=1}^{K} w_o(i,m,n,k)\cdot o_p(k,m,n) \tag{3.3}$$

where $\mathbf{W_o}$ is the 4 dimensional matrix of size (M by M_o by N_o by K), which connects hidden unit activation outputs or features to the output layer net vector $\mathbf{n}_{po}$, $\mathbf{o}_p$ is a 3 dimensional hidden unit activation output matrix of size ($M_o \cdot N_o \cdot K$) and $\mathbf{t_o}$ is the vector of biases added to net output function as in equation (3.3).

Before calculating the final error, the vector $\mathbf{n}_{po}$ is passed through an activation function such as softmax in 2.8. Finally the cross entropy loss function is calculated using equation 2.9.

3.2 CNN Training Using SD

Before training begins, a user decides between an approximation or classification model[65][66]. After deciding the model type, an objective function is chosen. Then the objective function is reduced with respect to the unknown weights. In this chapter, we discuss CNN training of classification models. We minimise the loss function E_{ce} using steepest descent.

Backpropagation([24], [25], [26], [27]) is a common method for calculating CNN gradients in steepest descent. The negative gradient of E_{ce} with respect to the output weight matrix $\mathbf{W_o}$ is written as,

$$\mathbf{G_o} = -\frac{\partial E_{ce}}{\partial \mathbf{W_o}} \tag{3.4}$$

The negative gradient of E_{ce} with respect to the input weight matrix $\mathbf{W_f}$ is written as,

$$\mathbf{G_f} = -\frac{\partial E_{ce}}{\partial \mathbf{W_f}} \tag{3.5}$$

The filter and output weights are updated as

$$\mathbf{W_o} = \mathbf{W_o} + z \cdot \mathbf{G_o} \tag{3.6}$$

$$\mathbf{W_f} = \mathbf{W_f} + z \cdot \mathbf{G_f} \tag{3.7}$$

where z is the learning factor. Here we use Adams optimizer [67] to find z and update weights.This optimizer is used in most CNN weight training algorithms. It is computationally efficient and easier to implement than optimal learning factor [58]. It uses momentum and adaptive learning rates to converge faster, which is said to be inherited from RMSProp[68] and AdaGrad[69]. The default parameters are given in [67].

3.3 Multi Layer CNN Notation

One layer CNNs are rarely used in real world applications. Multilayer CNNs with pooling[36][37][70], batch normalization[71], and dropout[72] layers are widely used. In this section, we describe a notation for multilayer CNNs.

Here, we keep the notation for inputs and outputs explained in section 3.1 as $\mathbf{F}$ for input image and $\mathbf{y}$ as the final output discriminant vector of the network. Let the number of layers be denoted as N_L, each layer layer consists of convolutional layers, activation layers. Each CNN can also include batch normalization, maxpool and dropout layer which is not shown in the notation. This also means that the start of every new layer will be a convolution layer. Now suppose if we have a multi channel image with elements $f(N_r, N_c, C)$, were $m = 1 : N_r$, $n = 1 : N_c$ and $c = 1 : C$. The output net function matrix of the convolution layer is $\mathbf{N}_{pL}$ and its activation output matrix is denoted as $\mathbf{O}_{pL}$, where L is the current layer number and C for $L > 1$ is the number of filters used in the previous layer. For $N_L = 3$, the net function output for the N_L^{th} layer is defined as

$$n_{pN_L}(k_{N_L}, i, j) = t_{N_L}(k_{N_L}) + \sum_{m=1}^{N_f}\sum_{n=1}^{N_f}\sum_{c=1}^{C_{N_L}} w_{N_L}(k_{N_L}, m, n, c) \cdot f_{N_L}(m + (i-1)s, n + (j-1)s, c_{N_L}) \tag{3.8}$$

We discuss the notation for $N_L = 3$ in detail. For p patterns and k_1 filter for the first hidden layer, the net function output for i^{th} row and j^{th} column is

$$n_{p1}(k_1, i, j) = t_1(k_1) + \sum_{m=1}^{N_f}\sum_{n=1}^{N_f}\sum_{c=1}^{C} w_1(k_1, m, n, c) \cdot f_1(m + (i-1)s, n + (j-1)s, c) \tag{3.9}$$

The activation output of the above equation would be

$$o_{p1}(k_1, i, j) = f(n_1(k_1, i, j)) \tag{3.10}$$

where, $o_1(k, i, j)$ is the k_1^{th} filter's first hidden unit activation output for the i^{th} row, j^{th} column of size (K by N_{rb} by N_{cb}).

For the second layer, the net function output for k_2 filters are found as

$$n_{p2}(k_2, i, j) = t_2(k_2) + \sum_{m=1}^{N_f}\sum_{n=1}^{N_f}\sum_{c=1}^{k_1} w_2(k_2, m, n, k_1) \cdot o_1(m + (i-1)s, n + (j-1)s, k_1) \tag{3.11}$$

After passing the second layer net function through its activations, we get $o_{p2}(k, i, j)$. Similarly for the third layer, the net function output will be calculated as follows

$$n_{p3}(k_3, i, j) = t_3(k_3) + \sum_{m=1}^{N_f}\sum_{n=1}^{N_f}\sum_{c=1}^{k_2} w_3(k_3, m, n, k_1) \cdot o_2(m + (i-1)s, n + (j-1)s, k_2) \tag{3.12}$$

In classification, the final layer's activation is often softmax cross entropy as defined in 2.9, but sigmoid activations can also be used. The advantage of softmax over sigmoid is that, the sum of all of the output values equal to 1, hence it is often assumed to represent probabilities.

Chapter 4

Shift Invariance in CNNs

In this chapter, we first define two types of show shift invariance in a CNN. We also describe some conventional methods for introducing shift invariance in a CNN. We then demonstrate unexpected shift invariance for shallow CNN. Finally we discuss some of the problems faced by shallow CNNs.

4.1 Classical Shift Invariance

Definition : Assume that a convolution layer processes input image, F with elements $f(m, n, c)$ producing relu activation outputs $o(m, n, c)$, where all elements less than zero are truncated to zero and c is the number of filters in the convolution layer. The system has classical shift invariance(CSI) if a shifted input image $F_{m_o,n_o,c}$ with elements $f(m-m_o, n-n_o, c)$ produces a shifted output image with elements $o(m-m_o, n-n_o, c)$.

A CNN's feature and input layers have CSI if the following conditions are met

(C1) All strides are equal to 1, so that no sub-sampling takes place

(C2) Objects in the input image are smaller than N_r by N_c

(C3) Shifts must be small enough that the object in each layer is not partially truncated by the edges of the image.

(C4) Input images contain the object to be shifted but no non-zero noise in which pixels surrounding the object are nonzero.

Note that nonlinear activations in each layer have no effect on shift invariance. If the input F is shifted by m_o,n_o, the net function output $n_{pL}(k_L, m, n)$ in layer L also shifts by (m_o, n_o), producing

$n_{pL}(k_L, m - m_o, n - n_o)$ and its activations $f(n_{pL}(k_L, m, n))$ are also shifted by (m_o, n_o), producing $o_{pL}(k_L, m - m_o, n_o)$.

Example : We demonstrate CSI for a CNN's early layers in figure 4.1, where the object inside the 16 x 14 original image is of size 5 x 5. This image is convolved with a filter of size 3 x 3 and the convolution ouput image is of size 14 x 12. Here, we use a stride of 1 with no padding of zeros around the border of the original 16 x 14 image. We observe that after convolution the features are in the left top as the original image indicated in green pixels.

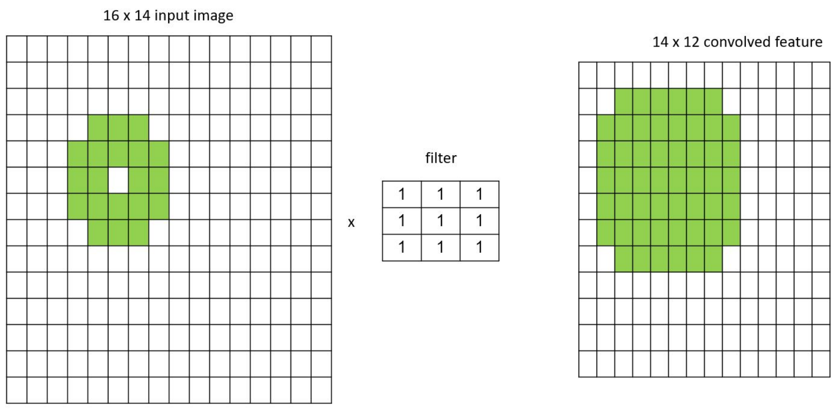


Figure 4.1: Convolution of 16 x 14 image with 3 x 3 filter

Suppose the object inside the original image is shifted to the opposite location in a 16 x 14 image as shown in figure 4.2. After convolution with the same filter used before, we obtain the features in the bottom right of the convolved image. Thus we demonstrate shift invariance in convolutional layer.

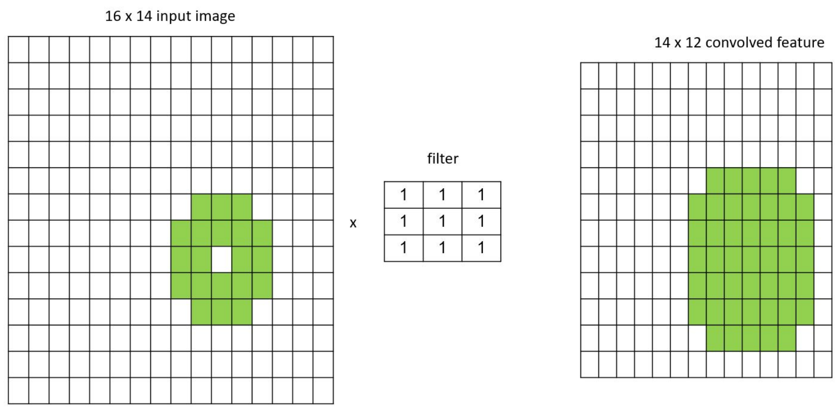


Figure 4.2: Convolution of 16 x 14 image with 3 x 3 filter with shifted image

4.2 Discriminant Shift Invariance(DSI)

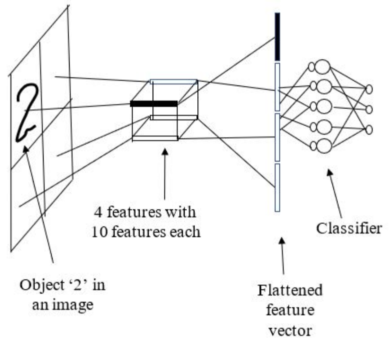
CNNs are assumed to be shift invariant neural networks based on the shared-weight architecture[8][9]. However recent work has shown that CNN-based classifiers are not always shift invariant[[73][74]].

Definition Let $d_i(\mathbf{F})$ denote i^{th} class output discriminant of a CNN with input $\mathbf{F}$ and let $d_i(\mathbf{F}_{m_o,n_o,c})$ denote the CNNs output discriminants for input $\mathbf{F}_{m_o,n_o,c}$. The model has DSI if,

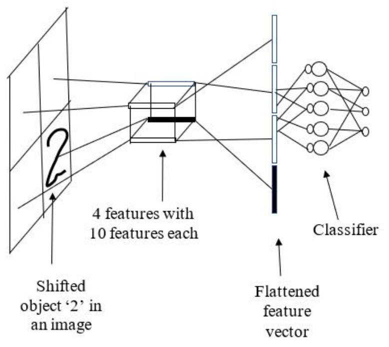
$$d_i(\mathbf{F}_{m_o,n_o}) = d_i(\mathbf{F}) \tag{4.1}$$

DSI means that the output discriminants should be unchanged when the input image f is shifted by any degree $m - m_o$ and $n - n_o$. This means that the classifier makes same predictions even if the object is shifted in the input image.

Example : Here, we demonstrate the output classifier in a CNN does not posses DSI using figure 4.3. We can observe that very few blocks deliver useful features to the classifier. As an object in the input image shifts horizontally or vertically, the useful features similarly shift position in the feature layer. The classifiers output discriminant $d_i(\mathbf{F}_{m_o,n_o})$ changes, which means that the classifier lacks DSI.



(a) Original Image with object in top left corner



(b) Original Image with object in bottom right corner

Figure 4.3: One Layer CNN Lacking DSI

figure 4.3a, shows an image that has an object '2' in the upper left quadrant. The dark segment contains the useful feature data with K = 10 and clear bars are noise. After flattening the image the second bar has the feature data associated with object '2'. Note that if the object '2' in the above image shifts to a different window, the dark segment containing useful features moves to a new location as shown in figure. 4.3b , damaging classifier performance. We can say that the MLP classifier does not have DSI. Hence the CNN as a whole lacks DSI.

To achieve DSI, we need to implement a shift removal method(SRM) as shown in figure (4.4) where the SRM goes somewhere between the final features layer and the output discriminants. In the figure, we observe that CNN achieves DSI even if the object inside the input image is shifted .

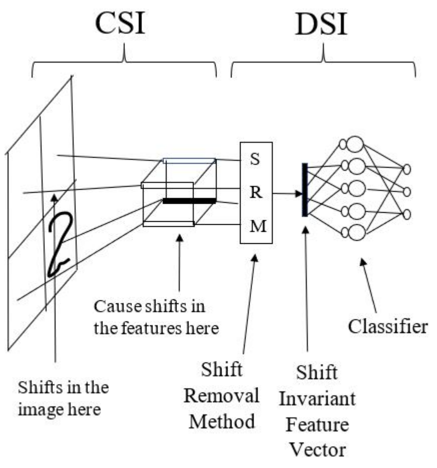


Figure 4.4: Shift Removal Example

4.3 Conventional Methods for Introducing DSI to CNNs

There are various conventional methods for generating shift invariance in CNNs. These methods remove shift in the feature layer. This involves discriminant layers to remove shift in input layer, feature layer or in the discriminant layer so that $d'(F_{m_o,n_o}) = d'_i(F)$, where d'_i represents $d_i(F)$ after the SRM has been introduced. The following are conventional SRMs.

(1) Global average pooling in the feature layer.

(2) Global max pooling in the feature layer.

(3) Layer by layer pooling.

(4) Capsule networks.

(5) Performing segmentation and processing segmented objects separately as in the RCNN.

4.3.1 Shift Invariance via Global Average Pooling

The lack of DSI described in subsection 4.2 can be corrected using the global average pooling(GAP) SRM[70]. Consider the *axial view* of a CNN feature layer below for an image with a small object. Two features, from two filters, are shown at each location.

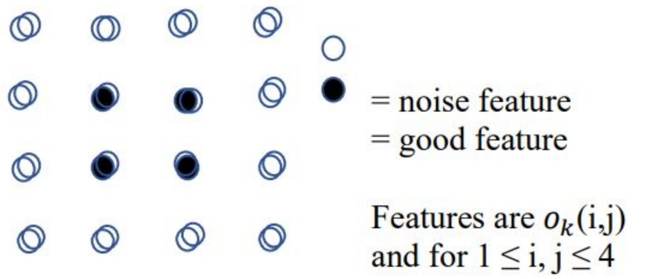


Figure 4.5: Axial View of a CNN Feature Matrix

The features for input block (i, j) are $o_k(i, j)$ for $k = 1$ to K, for K=2. Note that we can linearly average the blocks' features as

$$o_k = \frac{1}{M_o N_o} \sum_{i=1}^{M_o} \sum_{j=1}^{N_o} o_k(i,j) \tag{4.2}$$

leaving us with a total of K features which are now shift invariant. Figure (4.6) illustrates how the

GAP SRM introduces DSI.

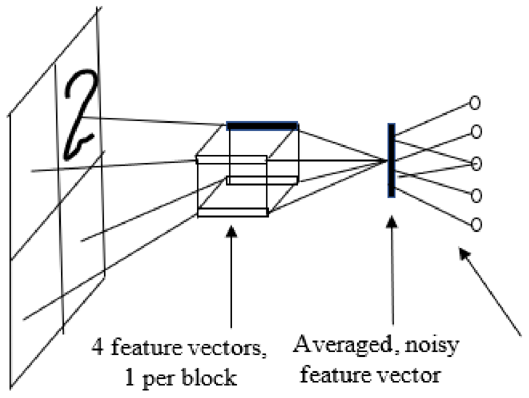


Figure 4.6: DSI CNN With GAP and Noisy Feature Vector

Similarly, figure (4.7) is a more typical depiction of global pooling in CNN feature extraction, where, the feature output of the trained CNN is denoted as $\mathbf{x}_p$ of size N_v by N_{fi}, where N_{fi} is the number of output flattened features. From the figure, $\mathbf{W}_{o2}$ is very sparse for global average pooling, where

$$\mathbf{W_o} = \mathbf{W_{o2}} \cdot \mathbf{W_{o1}} \tag{4.3}$$

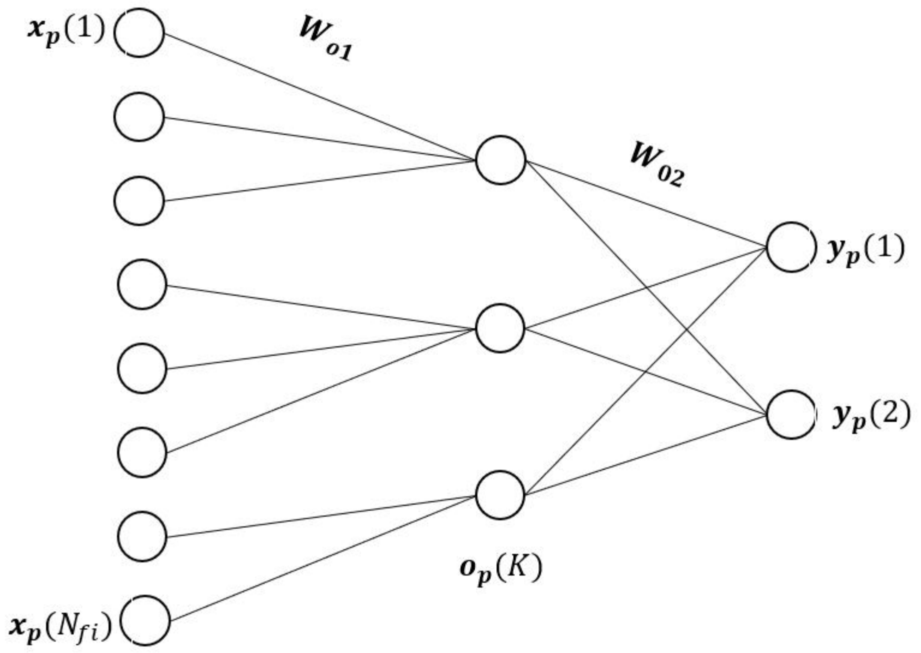


Figure 4.7: CNN With Global Average Pooling

Problems with Global Average Pooling(GAP) Averaging of indices by GAP in equation (4.3) introduces atleast two problems.

1. If block (i_1, j_1) contains the object to be classified, many of the remaining blocks have noise features $o_k(i, j) = e_k(i, j)$ for $(i, j) \neq (i_1, j_1)$, that degrade the performance of the k^{th} GAP feature as

$$o_k = o_k(i_1, j_1) + \sum_{(i,j)\neq(i_1,j_1)} e_k(i, j) \tag{4.4}$$

2. Even when there is no noise GAP distorts the useful features by averaging them.
   However, if useless image blocks are thresholded to zero, then

$$o_k = o_k(i, j_1) \tag{4.5}$$

and noise is not a problem. In general the network has a 4-dimensional output weight array so that the m^{th} class output net function is

$$n_{po}(m) = t_o(m) + \sum_{i=1}^{M_o}\sum_{j=1}^{N_o}\sum_{k=1}^{K} w_o(m,i,j,k)\cdot o_k(i,j) \tag{4.6}$$

Using the new features as shown in 4.3 we have

$$\begin{aligned} n_{po}(m) &= t_o(m) + \sum_{k=1}^{K} w_o(i,k)\cdot o_k \\ &= t_o(m) + \sum_{k=1}^{K} w_o(m,k)\sum_{i=1}^{M_o}\sum_{j=1}^{N_o} \cdot o_k(i,j) \\ &= t_o(m) + \sum_{k=1}^{K}\sum_{i=1}^{M_o}\sum_{j=1}^{N_o} w_o(m,k)\cdot o_k(i,j) \end{aligned} \tag{4.7}$$

where $w_o(i,j,m,k) = w_o(i,k)$. If useless blocks are not thresholded to zero, noise may damage performance.

Ex: A shallow CNN classifier is used for small objects that can be shifted. How can we enforce GAP?

Solution: First calculate weights $w_o(m,k)$ as

$$w_o(m,k) = \frac{1}{M_oN_o}\sum_{i=1}^{M_o}\sum_{j=1}^{N_o}\sum_{k=1}^{K} w_o(m,i,j,k) \tag{4.8}$$

Then fill up the 4-D weight array as $w_o(m,i,j,k) = w_o(i,k)$ for all m,i,j and k.

The CNN may approximately learn to do this during training, but direct implementation of the method can be tried.

Ex: In equation (4.8), feature vectors occurring at the edges of the image, such as $o_k(1,1)$ or $o_k(N_{rb},m)$ may be noisy or useless if the object to be recognized is never located there. This means that the averaging operation fails. A possible solution is a weighted average approach such as

$$o_k = \frac{1}{M_oN_o}\sum_{i=1}^{M_o}\sum_{j=1}^{N_o} c(j,m)o_k(i,j) \tag{4.9}$$

where,

$$c(i,j) = \sum_{m=1}^{M}\sum_{k=1}^{K} |w_o(m,i,j,k)| \tag{4.10}$$

4.3.2 Global Max Pooling

In this subsection, we discuss global max pooling, figure. 4.8 is a third depiction of shift invariance in CNN feature extraction

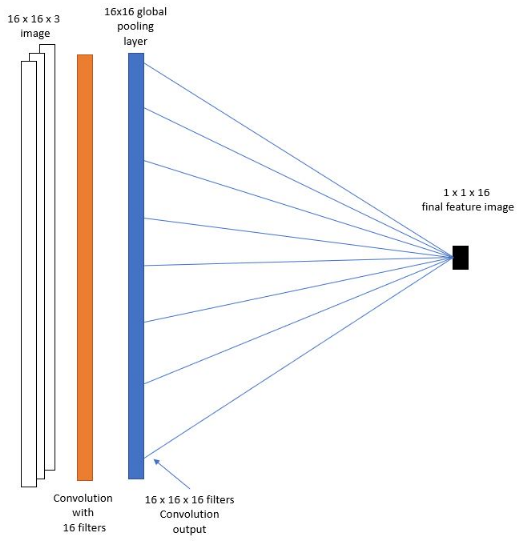


Figure 4.8: Global Max Pooling in CNN's Feature Extraction Layers

As we saw earlier, GAP involves averaging the feature matrix. Global max pooling(GMP) involves finding the maximum pixel for each pixel in the feature matrix. From figure 4.8 we see that a 16 x 16 input image is convolved using 16 filters and a global pooling layer, whose size is equal to input image, i.e 16 x 16 is used to down - sample the image to (1 x 1 x K). These layers are often used to replace the fully connected layers where the total number of filters in the final feature layer are equal to the number of classes.

4.3.3 Layer By Layer Pooling

In this subsection, we show how layer by layer pooling can be derived from global pooling. If figure (4.8) represents global max pooling(GMP), we can decompose the network as shown in figure 4.9. Instead of having a single pool layer of the size of the image, we spread out the pool layers until we get a final feature image.

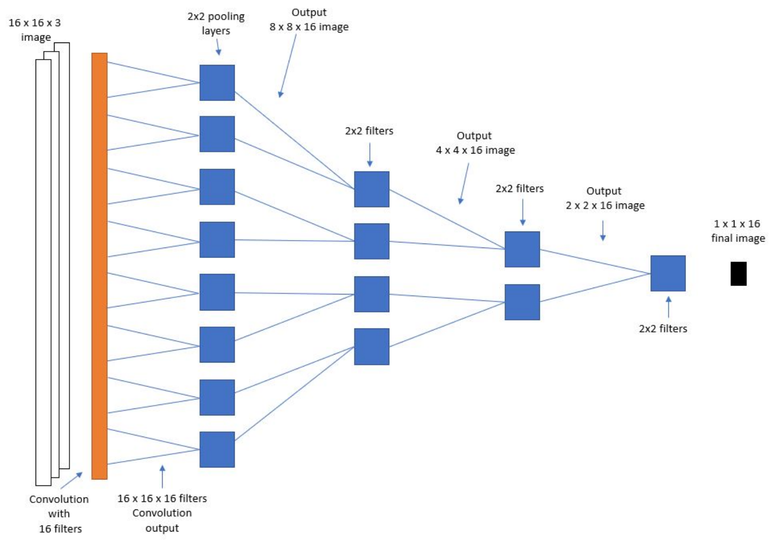


Figure 4.9: Multiple Pooling in CNNs Feature Extraction Layers

In figure 4.9, we assume that the input image is that of figure 4.8. Here we accomplish global max pooling by breaking it up into layers of smaller maxpool units and the final layer feature output of size 1x1x16, where 16 is the total number of filters used in the convolution layer. One problem with figure 4.9 is that we don't have the flexibility to change the number of filters in the pooling layers. To change the number of filters or features, we insert convolution layers in between the maxpool layers as seen in figure 4.10. The advantage of this approach is that we can get more high level features on the final layers while preserving DSI.

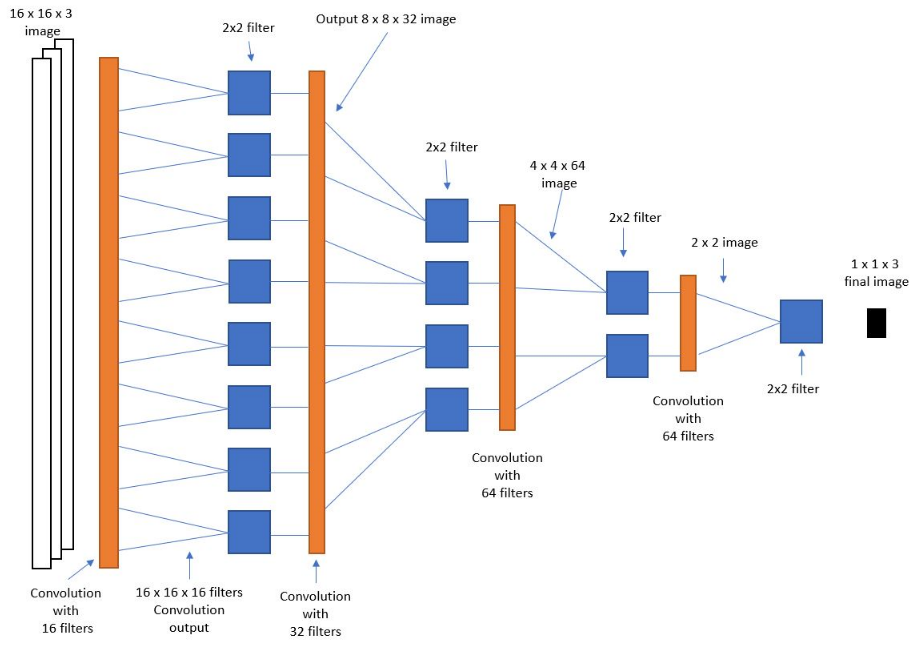

16 x 16 x 3 image
2x2 filter
Output 8 x 8 x 32 image
2x2 filter
4 x 4 x 64 image
2x2 filter
2 x 2 image
1 x 1 x 3 final image
2x2 filter
Convolution with 64 filters
Convolution with 64 filters
Convolution with 16 filters
16 x 16 x 16 filters Convolution output
Convolution with 32 filters

Figure 4.10: Shift Invariant CNN With Multiple Pooling with Convolution layer in between

A similar pooling approach is explained in [75] which uses a well known signal processing anti-aliasing technique called blurpool, where low-pass filtering is done before down-sampling with max-pooling. The author showed promising results on commonly used deep learning architectures[76][77].

Most recent deep learning architectures use the above approach instead of GMP so that the final feature layer is of size (1 x 1 x K), where K is defined as number of filters. One such deep learning architecture is inception-resnet-v2[78], a type of convolutional neural network with 164 layers, deep trained on the ImageNet database[79]. The network has input image size 299 by 299. A downsampling technique such as average pooling is used to force the final feature layer to size (1 x 1 x 1536), where 1536 is the number of filters in the last convolution layer. Similarly, in deep learning architecture densenet201[76] a similar downsampling technique is used to force the final feature layer to be of size (1 x 1 x 1920), where 1920 is the number of filters concatenated in the final feature layer. Similar downsampling techniques are used in resnet50[77] architecture, Xception[80] architecture and in the mobilenetv2[81] architecture.

4.3.4 Capsule Networks

The downsampling pooling methods described earlier, ignore the sampling theorem[75] and fail to learn the precise spatial relationship between higher level parts such as nose and mouth. *Geoffery*

Hinton has consequently said that *'The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster'*[82]. For example CNNs are successful in recognizing different elements of a face in an image such as eyes nose mouth, but are not sensitive to the arrangement of the features on a given face, meaning that the nose is in between 2 symmetric eyes and above the mouth, etc.

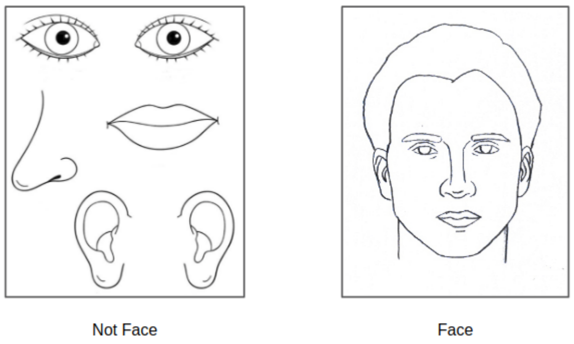


Figure 4.11: CNN face recognition problems

This problem is illustrated in figure 4.11[83]. To solve the shift insensitivity problem explained above, the fourth approach is a new type of neural network called a capsule network[[84] [83]].The author has attempted to more closely mimic neural biology. The capsule network implements groups of neurons that encode spatial information as well as the probability of the an object being present[83]. Traditional neural networks look for only features in images which are obtained by convolution. Capsule networks consist of primary capsule layer, higher capsule layers and loss calculations.

Primary capsule layer consists of inverse graphics, where inverse graphics is a reverse process of finding and breaking down different objects and its parameters from the image. It consists of small group of neurons called as capsules. The main task of capsule is to predict the presence of an object and its parameters at a specific location. Capsules are vectors that represents the object and its likelihood. Each capsules represent vector of each object and its variation, such as eye position, size, oriention etc. Similarly, a second capusule can have vector representation of nose position, size, orientation etc. This is achieved by convolution, reshape and a squash function, where the squash function is an activation function used to normalize the magnitude of vectors. This information is further sent to the higher capsule layer which can represent the whole face. This transfer step is done by setting up routing weights[83]. Routing weights is a dynamic routing algorithm proposed in[83] that enables transfer of information between primary and higher capsule layer. Finally, the information is further fed through softmax function for prediction by calculating margin loss and also an image is being reconstructed using multiple fully connected layers.These networks are said to preserve the hierarchical pose relationships between different parts of an object and only a fraction amount of data is needed for capsule networks to achieve state of art results[84] [83]. Capsule networks are still in a research phase and practical applications are still pending.

4.3.5 CNNs with Segmentation

Another process to introduce shift invariance is to perform segmentation and process each segmented object separately. This is still an active area of research. Segmentation algorithms have bloomed in recent years with the development of the first real time object detection algorithm called Region-Based CNN(RCNN) in 2014. RCNN uses a selective search method to extract 2000 region proposals from a particular image. Selective search[85][86] is a recursive greedy algorithm that combines similar regions to generate final region proposals. These region proposals are then fed through a pretrained CNN which was a modified version of Alexnet[87] to extract features of the proposed regions . These extracted features are finally fed into an SVM to classify each object in the proposed region. RCNN also predicts bounding box values for the object using bounding box regressor. Similar algorithms[88][89][90][91][92] were published which improve accuracy and processing time. These algorithms are mostly used for real time object detection.

4.4 Unexplained DSI in Shallow CNN

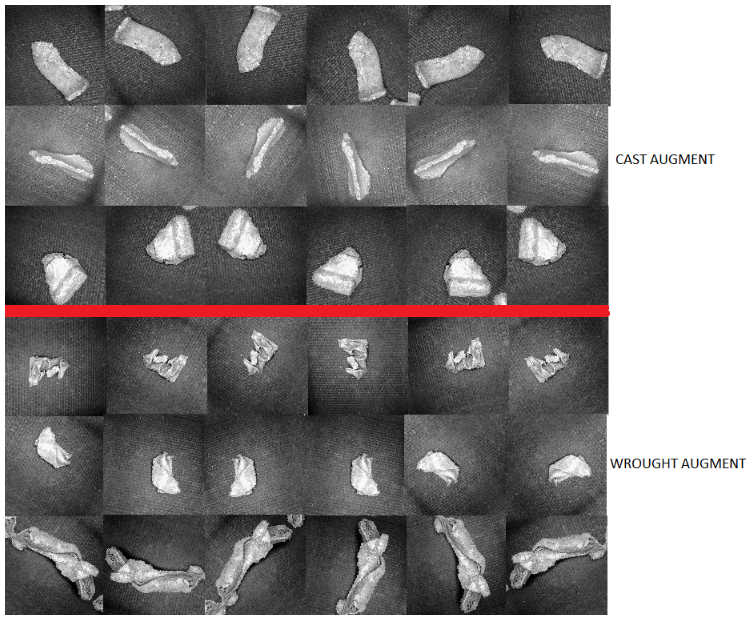


Figure 4.12: Scrap data example

Consider an aluminum scrap recognition example from a real world application where the scrap is to be recognized as cast or forged, based upon color images having $N_r = 128$ and $N_c = 128$. The images used as part of the training and testing process are down sampled to $N_r = 32$ and $N_c = 32$. There is one piece of scrap per image of a size much smaller than N_r by N_c . Also the pieces of scrap can be shifted anywhere inside its image. figure 4.12 gives examples of the scrap dataset where the first three rows and six columns are cast images and the remaining three rows and six columns are forged images. We observe that the images can be shifted and rotated. We built shallow CNNs for the this dataset. For all networks, filters are of size 3 x 3. The first shallow CNN has L convolutional layers with relu activations where L = 1. It also has a linear classifier trained using the softmax cross-entropy loss function E_{ce}. The second shallow CNN is similar except that it has a maxpooling layer added after the relu activations. CNN number 3 has the same structure as CNN number 1 except that it has an additional convolutional layer with relu activations inserted so that L = 2. Similarly CNN number 4 starts with the structure of CNN number 2 except that it has an extra convolutional layer with relu activations along-with maxpooling layer. CNN number 4 has the same structure as CNN number 1 except that it has a linear MLP instead of a linear classifier. The five networks above are implemented in MATLAB using Adams optimizer[67] with an initial learning rate of 0.001. The training data was randomly read in batches of 32 images from the training set. The results are given in Table 4.1

Network No.	Testing accuracy	CNN layers(L)	Number of filters (K)	Maxpool included
CNN 1	85.31	1	5	No
CNN 2	84.20	1	5	Yes
CNN 1	91.07	1	64	No
CNN 2	89.90	1	64	Yes
CNN 5	90.35	1	64	No
CNN 3	88.26	2	5	No
CNN 4	86.48	2	5	Yes
CNN 3	95.24	2	64	No
CNN 4	94.88	2	64	Yes

Table 4.1: Results for 5 filters for scrap data

Most investigators assume that DSI shift invariance must be accomplished by layer by layer pooling as max or average pooling has an advantage of either summing[36] or averaging[37] blocks of pixels. From Table 4.1 we observe that

(1) CNNs with layer by layer maxpooling don't perform as well as the CNNs without maxpooling.

(2) Shallow CNNs with no pooling of any kind may have DSI

Hence we can say that the cause of DSI in shallow networks is unknown. There are several other problems associated with shallow CNNs which we discuss later for these examples.

Chapter 5

Piecewise Linear activations(PWL)

Activations functions play an important role as these are used to provide non linearity to the network. Relu activations have found huge success in many fields[39]. However, relu's are not optimal for all applications and are not used in all networks[93][44]. Compact optimal CNNs have a very different activations for each filter. PWL activations have universal approximation but relu and leaky relu do not. Piecewise linear activations are an active area of research. These PWL activations are multiple relu functions.

In this subsection, we briefly describe exsisting PWL activations and lay a theoretical foundation for adaptive PWL activations.

5.1 Fixed PWL Activations

PWL functions are composed of Relu activations [39]. Activations such as sigmoid[40] and tanh[41] can be approximated using relu units. Several investigators have tried adaptive PWL activation functions in MLPs[45] and deep learning[49] and have published promising results. One of them includes hybrid piecewise linear units(PLU) with an activation function that is a combination of tanh and relu activations[44].

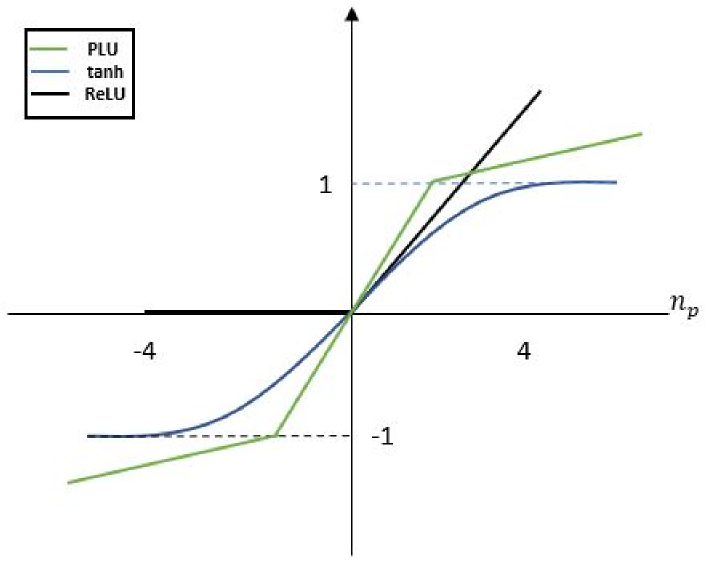


Figure 5.1: Fixed PWL activations

From figure 5.1, we see that PLU is a combination of relu and tanh activations. The equation for calculating fixed PWL activations is given as

$$o_p(k) = max\left[\alpha(n_p(k)+c)-c, min(\alpha(n_p(k)+c)-c, n_p(k))\right] \tag{5.1}$$

where, α and c are user chosen parameters. The author has also proposed that the α can also be a trainable parameter. The paper [44] demonstrates the performance of fixed PWL in an MLP for paramteric functions, 3D surface approximation and invertible network datasets. The author shows that fixed PLUs work better than relu functions as PLUs are represented using more hinges than relu functions. The author also shown promising results using CNN for the cifar 10[50] dataset. The fixed PWL activation function has only 3 linear segments, hinges $H = 3$ and will not be adaptive until the α parameter is trained in every iteration. Since the H is fixed and there is minimal or no training , there is no universal approximation.

5.2 Adaptive PWL activations

An alternate piecewise linear activation has been demonstrated[49], which is specifically designed for deep networks with trainable PWL activations. This method therefore outperforms the fixed PWL activation in section 5.1. The adaptive PWL activations here can equal those of section 5.1 and can also generate more complicated curves. The author has implemented an adaptive piecewise linear activation unit where the number of hinges H is a hyperparameter that is user chosen. The author shows the best results for CIFAR10 data using $H = 5$ and $H = 2$, and for CIFAR100 data using $H = 2$ and $H = 1$(no activation hinge training). The initialization for there adaptive activations is not properly specified.

The equation for calculating the the adaptive activation is given as

$$o_p(k) = max(0, n_p(k)) + \sum_{s=1}^{H} a_k^s \cdot max(0, -n_p(k) + b_k^s) \tag{5.2}$$

where, a_i^s and b_i^s for $i = 1..H$ are learned using gradient descent. a_i variables control the slopes of the linear segments and b_i^s determine the locations of sample points.

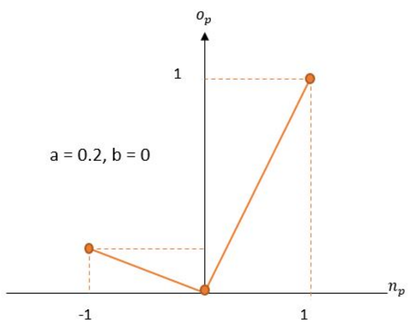


Figure 5.2: 2 relu curves

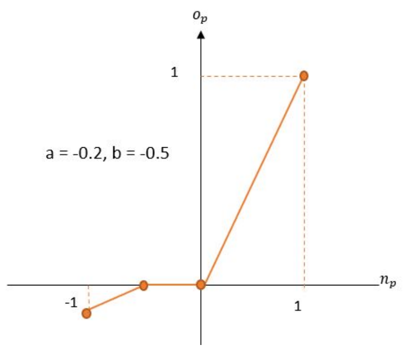


Figure 5.3: 4 relu curves

figure 5.2 shows adaptive PWL with slope a as 0.2 and b as 0. Similarly, figure 5.3 shows adaptive PWL with slope a as -0.2 and b as -0.5.

5.3 An Alternate PWL Method

Section 5.2 describes an adaptive PWL activation which trains the locations and slopes of the hinges. The author claims that a small number of hinges achieved better results. We see in section 5.1 that a network with two hinges outperforms relu for a particular application. To approximate a linear output, only one hinge is needed on the PWL curve. Similarly, to approximate a quadratic output, the number of hinge sets on the PWL curve should be larger than three. Therefore, the number of hinges should not be less for more complicated datasets. This can also result in using fewer hidden layers and filters as the network doesn't need to train for a longer time.

We further investigate the use of PWL activations in CNNs[94] and we first determine that PWL activations can approximate any other existing activation functions. Consider a CNN filter's net function n_1 defined as

$$n_1 = t + \sum_{m=1}^{N} w_i(m) \cdot x(m) \tag{5.3}$$

where t is the threshold, $w_i(m)$ is the m^{th} filter weight, and $x(m)$ is the m^{th} input to the net function. The filter can be represented as $\{\mathbf{w_i}, t\}$. The continuous PWL activation $f(n_1)$ is

$$f(n_1) = \sum_{k=1}^{N_s} a_k \cdot r(n_1 - ns_k) \tag{5.4}$$

where N_s denotes the number of segments in the PWL curve, $r()$ denotes a ramp (relu) activation, and ns_k is the net function value at which the k^{th} ramp switches on.

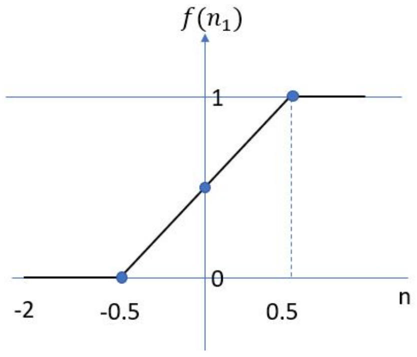


Figure 5.4: Approximate sigmoid using 2 relu curves

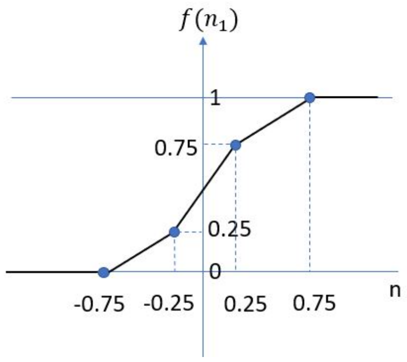


Figure 5.5: Approximate sigmoid using 4 relu curves

Figures 5.4 and 5.5 show approximate sigmoid curves generated using relu activations where figure 5.4 has $N_s = 2$ relu curves and figure 5.5 has $N_s = 4$ relu curves. Comparing the two figures we see that larger values of N_s lead to better approximation. The contribution of $f(n_1)$ to the j^{th} net function $n_2(j)$ in the following layer is

$$+n_2(j) = f(n_1) \cdot w_o(j) \tag{5.5}$$

Decomposing the PWL activation into its N_s components, we can write

$$\begin{aligned} +n_2(j) &= \sum_{k=1}^{N_s} a_k \cdot r(n_1 - ns_k) \cdot w_o(j) \\ &= \sum_{k=1}^{N_s} w_o'(j,k) \cdot r(n_1(k)) \end{aligned} \tag{5.6}$$

where $w'_o(j,k)$ is $a_k \cdot w_o(j)$ and $n_1(k)$ is $n_1 – ns_k$.
A single PWL activation for filter $\{\mathbf{w_i}, t\}$ has now become N_s relu activations $f(n_1(k))$ for N_s filters, where each ramp $r(n_1 - d_k)$, is the activation output of a filter. These N_s filters are identical except for their thresholds. Although, relu activations are efficiently computed, they have the disadvatage that back-propagating through the network activates a relu unit only when the net values are positive and zero, this leads to problems such as dead neurons[95] which means if a neuron is not activated initially or during training it is deactivated. This means it will never turn on causing gradients to be zero leading to no training of weights, Such relu units are called dying relu[96].

An adaptive PWL activation was defined for a MLP [94], where the initial PWL activation was derived from the sigmoid activation. The PWL hinges used are fixed and range from -4 to $+4$. Below we show the performance for the PWL activation using a sinusoidal example to demonstrate the power of our adaptive PWL activations.

Suppose we have input data, which are random numbers between 0 and 4π, and the output is the sine of the inputs. The plot for input vs target is figure 5.6

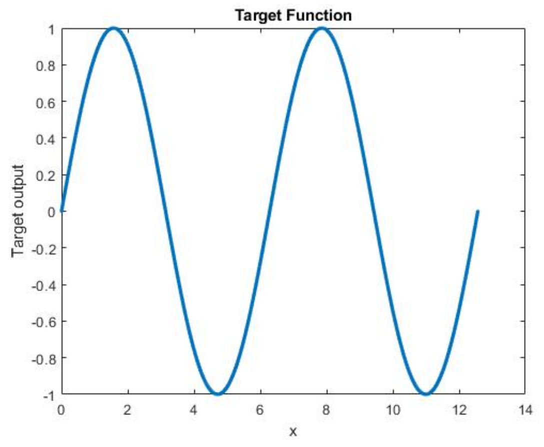


Figure 5.6: Sine Training Data

For this experiment, we use a MLP with one hidden layer and two hidden units and 10 iterations. We train four different networks. Each network has one of four different activation functions, relu, tanh, sigmoid and our designed adaptive activation. All four networks have the same structure and the same weight initialization except for the activations. We use SD method with optimal learning factor[58] for each network. For the adaptive PWL activation, we use $H = 9$ hinges.

figure 5.7 shows the original target and predicted target plot using relu activation. We observe that that relu activation does not approximate the sine output. figure 5.8 shows the original target and

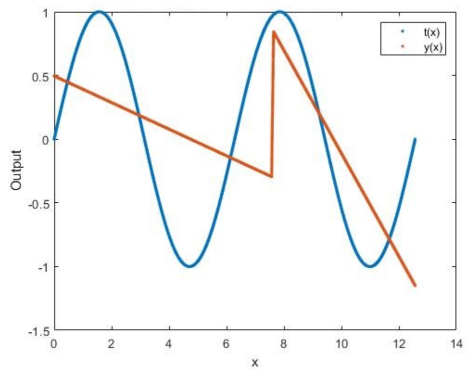


Figure 5.7: t(x) and y(x) for relu activations

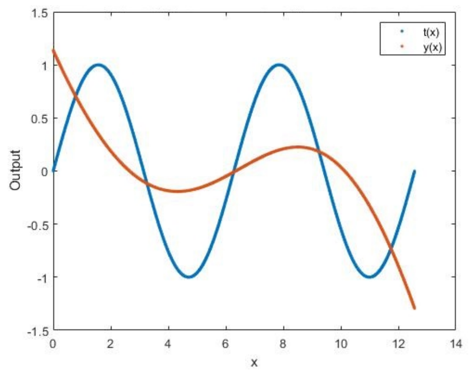


Figure 5.8: t(x) and y(x) for sigmoid activations

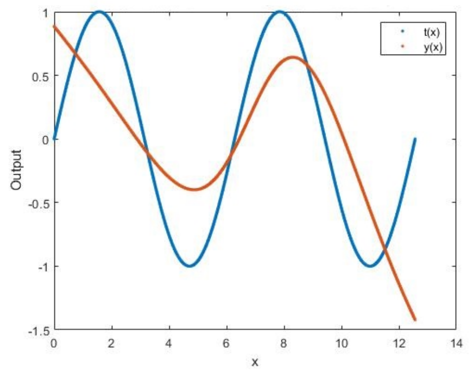


Figure 5.9: t(x) and y(x) for tanh activations

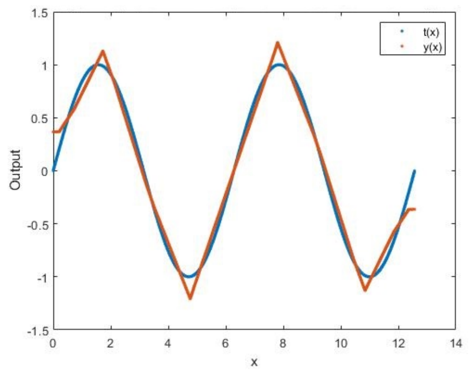


Figure 5.10: t(x) and y(x) for PWL activations

predicted target plot using sigmoid activation. We can observe that that sigmoid activations do not approximate the sine output but it performs better than relu activations. figure 5.9 shows the original target and predicted target plot using tanh activation. We can observe that tanh activations approximates the sine output better than relu and sigmoid activations. Finally, figure 5.10 shows the original target and predicted target plot using adaptive PWL activation. We observe that adaptive PWL activations approximate the sine output using 10 iterations. We observe that uisng PWL activations the predicted output has no curves at the top and bottom part of positive and negative half cycle. We suspect that this is because is because of the linear units we use in PWL activations. We verify this by increasing the number of hinges from 9 to 20 and 30, with same network structure and parameters and check if the we get a better target output.

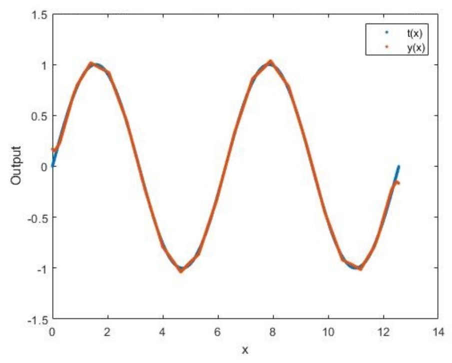


Figure 5.11: t(x) and y(x) using 20 PWL Hinges

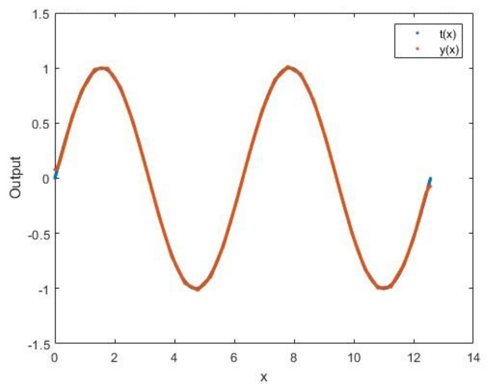


Figure 5.12: t(x) and y(x) using 30 PWL Hinges

From figure 5.11, we observe that as we increase H, the network approximates a sinusoid better. More complicated target functions require larger values of H.

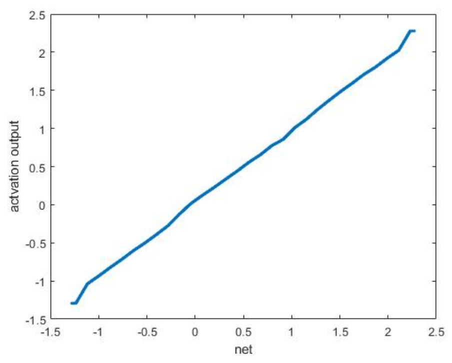


Figure 5.13: Activation vs Net for 1^{st} hidden unit for 30 hinges

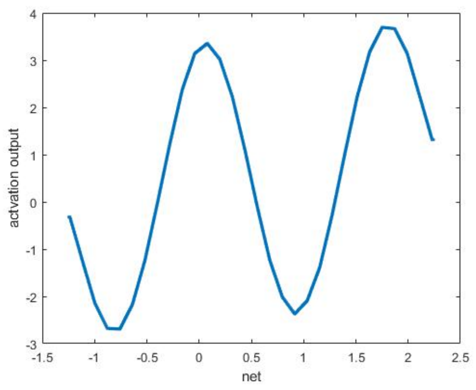


Figure 5.14: Activation vs Net for 2^{nd} hidden unit for 30 hinges

Figure 5.13 and 5.14 shows the net vs activation output plot for the 1^{st} and the 2^{nd} hidden unit for PWL activations with 30 hinges. From the plots we can observe that the first hidden unit curve is linear and the second hidden unit curve is the almost a mirror image of the output we are trying to predict. We can also assume that the network with PWL activation just uses one of the hidden unit and can hence be trained with a smaller number of hidden units.

Chapter 6

Problems and Completed Work

In this chapter, we list some problems associated with shift invariance and PWL activations. We also give our goals for this dissertation and show all completed tasks related to the goals.

In the shift invariance review of chapter 4, we observe that CSI is achieved using convolution layers but DSI is usually not achieved without an SRM technique. From chapter 5, we observe that we have good generalization for sinusoidal data using PWL activations in an MLP. The designed PWL adaptive activations in section 5.3 suffer from a problem of overfitting as the number of hinges is increased which can be seen from the results in [94]. This can be due to traditional weight initialization methods, initialization of the PWL hinges and number of hinges. PWL activations also face several problems during training. Below, we describe some of these problems.

P(1) It is unclear how shift invariance is achieved in shallow CNNs that lack pooling. Movements of an object in the input image cause similar movements of features in the final relu layer so DSI should be absent.

P(2) It is not clear how CNN output weight matrices can be analyzed to detect the presence of DSI.

P(3) As CNNs are not optimal as shown in subsection 4.4 and 5. It is not clear if background noise in an image affects training for GAP and other cases.

P(4) Activations such as relu and leaky relu are oversimplified PWL activation functions which lack universal approximation.

P(5) It is unclear how to train CNNs containing PWL activations.

P(6) It is unclear whether or not PWL CNNs train faster or perform better than ReLU activation CNNs.

P(7) It is not clear how to grow or prune unnecessary hinges in PWL CNNs.

6.1 Completed Goals and Tasks

In this section, we show goals and tasks for this dissertation. Our goals are to (1) investigate shift invariance in shallow CNNs and (2) investigate the performance of PWL CNNs. The specific tasks are to:

T(1) Develop a measure of DSI and use it to determine which CNNs have DSI.

T(2) Determine if shallow CNNs have learned to implement GAP. If not, analyze their output weight matrices using section 4.3.1.

T(3) Vary the background noise level and see how this affects CNN performance.

T(4) Develop efficient intialization algorithms for hinges in a PWL CNN.

T(5) Develop efficient training algorithms for PWL CNNs in MATLAB and Pytorch.

T(6) Determine whether or not PWL CNNs require fewer filters.

T(7) Implement a method to grow or prune hinges in shallow PWL CNNs.

Chapter 7

Shift invariance Work

In this chapter, we first describe a measure of DSI for trained CNNs. We then apply this measure to several shallows CNNs.

7.1 Measuring DSI

In this section, we describe a measure of DSI in shallow CNNs and then apply the measure to output discriminants with and without GAP. For this purpose, we use shifted versions of original image and calculate the shift invariance(SI) measure for the output discriminants for shifted and original images. This measure can be applied to any CNN, with any or no DSI.

To demonstrate the measure of DSI, we shift an original image. We then pass the original and shifted image in multiple convolution layer and then compare the output discriminant of the last convolution layer of the original image with the output discriminant of the last convolution layer of each of the shifted images. We show examples of one of each class for scrap data.

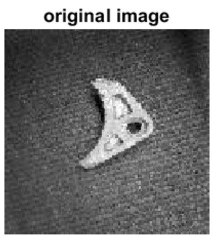
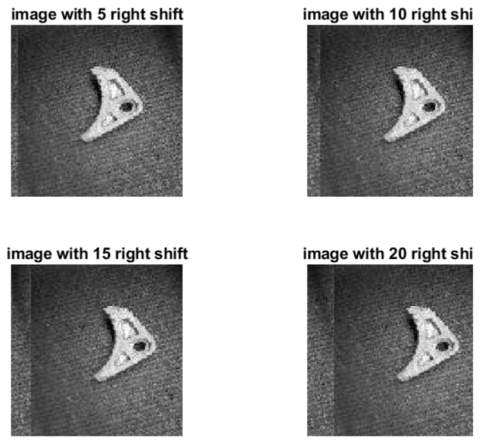
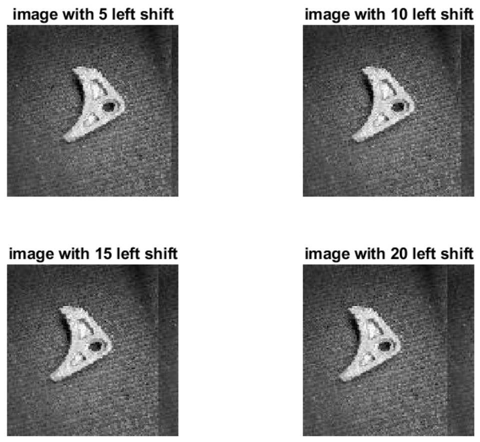


Figure 7.1: Original cast611 image



(a) Right shifted versions of cast611 image



(b) Left shifted versions of cast611 image

Figure 7.2: Shifted versions of cast611 image

We first show an example of cast(class one) imagefigure 7.1 is the original cast image and figure 7.2 are the shift images of cast images, where figure 7.2a are the right shifted version of cast images and figure 7.2b are the left shifted version of cast images. Similarly, we show an example of wrought(class two) image, were figure 7.3 is the original wrought image and figure 7.4 are the shift images of cast images, where figure 7.4a are the right shifted version of cast images and figure 7.4b are the left shifted version of cast images

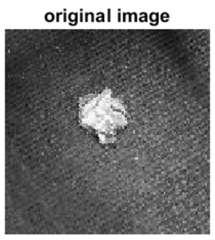
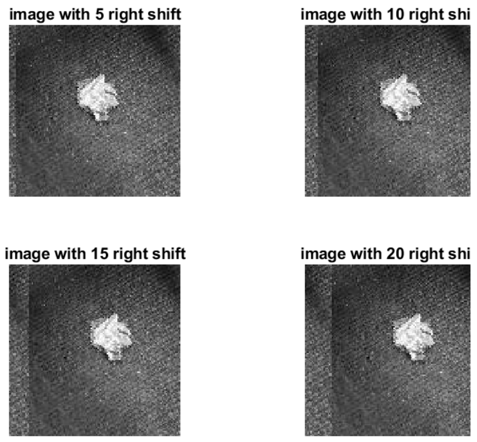
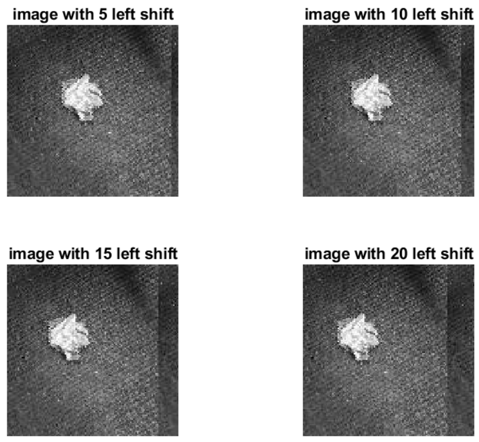


Figure 7.3: original wrought231515 image



(a) Right shifted versions of shifted wrought231515 image



(b) Left shifted versions of shifted wrought231515 image

Figure 7.4: Shifted versions of wrought231515 image

The SI measure of DSI is calculated using the Euclidean distance normalized by the sum of the norm of the vectors. We use normalization in case that one of the vectors is very small. Let $\mathbf{d}$ be the original image's output discriminant vector and $\mathbf{d}_s$ is the shifted image's output discriminant, where s is [right5, right10, right15, right20, left5, left10, left15, left20]. right5 means shifting image right by 5 pixels and left5 means shifting image left by 5 pixels.

$$SI = \frac{1}{T_s} \sum_s \frac{||d - d_s||^2}{||d||^2 + ||d_s||^2} \tag{7.1}$$

where, T_s is the total number of shifted images. To find the SI measure, we build a model which consists of 2,3,4 and 5 convolution layers and we measure the SI by inserting original image through three convolution layers and getting the final output discrimant. Similarly we do the same for shifted image. Finally, we calculate SI using equation 7.1. We show examples from each of the 2 classes. We first show the SI measure for Cast(class 1) image. We show the average SI of left shift and right shift of pixels $[5, 10, 15, 20]$ of the original image for GAP and non pooling output discriminants.

Convolution layers N$_L$	Activations output dicriminants	SI for averaged output discriminants
2	Original image vs shifted images	0.0010
3	Original image vs shifted images	0.0015
4	Original image vs shifted images	0.0021
5	Original image vs shifted images	0.0029

Table 7.1: SI for GAP

Convolution layers N$_L$	Activations output dicriminants	SI for all output discriminants
2	Original image vs shifted images	0.1986
3	Original image vs shifted images	0.2261
4	Original image vs shifted images	0.2836
5	Original image vs shifted images	0.3806

Table 7.2: SI without GAP

Table 7.1 shows the SI for GAP output discriminants for original image vs shifted images and Table 7.2 shows SI for non GAP output discriminants for original image vs shifted images. The results in above tables are averaged over images from different classes. From Table 7.1 and Table 7.2 we can observe that SI for GAP is very close to zero as expected and SI for non pooled output discriminants is significantly larger than 0. We also observe that as the number of convolution layers increases, the SI measure gets worse.

Now, as we have shown in section 7.1 that output disciminants of GAP for shifted images give us similar discriminant values. We check the performance of the GAP network. Consider the shallow CNN trained in the section (4.1.2). Here, we save the feature output of the trained CNN described in section (4.1.2) as $\mathbf{x}_p$, where the size of $\mathbf{x}_p$ is N_v by N_{fi}, where N_{fi} is the number of output flattened features, we also save the output layer net functions as $Y_p(i) = n_{po}(i)$. We train a network that consists of $\mathbf{x}_p$ as inputs and output as i_c. We minimize the weights using linear classifier with

MSE- OR[54][55] objective function. The structure of the algorithm is describe in the appendix. The testing accuracy is found on testing data by finding the best weights using cross validation. Below are the results for linear classifier for GAP features, weighted GAP(WGAP) features, concatenation of the average and weighted average(GAP-WGAP) features and all output discriminant(A) features in Table 7.3

Network	GAP features testing accuracy	WGAP features testing accuracy	GAP-WGAP features testing accuracy	Original features **x**
CNN1 5 filters	58.193	60.445	61.86	84.534
CNN2 5 filters	63.37	63.227	64.618	79.277
CNN1 64 filters	65.95	73.106	73.58	92.03
CNN2 64 filters	70.35	71.106	71.58	92.55
CNN3 5 filters	60.45	58.832	72.77	88.261
CNN4 5 filters	65.9	66.203	69.68	87.427
CNN3 64 filters	72.38	75.633	76.467	95.799
CNN4 64 filters	79.25	80.25	80.278	95.632

Table 7.3: Testing accuracy for Linear classifier on extracted features from output discriminants from CNN

Table 7.3 displays the linear classifier results for GAP, WGAP, GAP-WGAP and A features for the same CNN structures shown in subsection 4.4. Here, we see that results using A features gives the best testing accuracy. Furthermore, Comparing Tables 7.3 and Table 4.1 we observe that we get better testing accuracy for the CNN3 and CNN4 networks with 64 filters using linear classifier for A features which are highlighted in red in the table. We also can see that linear classifier using GAP features gives us the worst testing accuracy but the lowest SI measure as compared to A features. We can also see that we get better results for WGAP features. When we combine the GAP and the WGAP features and train the classifier we get better results than using GAP or WGAP alone. Hence, one should try concatenating different down sampled pool features during training.

We observed in section 7.1 that GAP gives DSI but give low testing accuracy. The possible reasons for low testing accuracy can be beacause pattern storage of W_o inn GAP is $(k+1)$, which is small and pattern storage of W_o without GAP is (N_{fi}), which is much larger. Therefore in the next section, we try a factored W_o with a larger pattern storage than that of GAP.

7.2 Modified Global Average Pooling

From table 7.3, we saw that GAP did not give us better testing accuracy and we gave a possible reason involving small pattern storage of GAP features as compared to Original features. We show a modified GAP where the W_{o1} weight in the sparse network in figure 4.7 is initialized and trained instead of using a constant as used in GAP.

From the sparse network shown in figure 4.7 we can see that each filter images are global average pooled, hence we have K is equal to the number of filters in the final layer feature layer. In Modified Global pooling, instead of multiplying the features output with one divided by total numbers of rows and columns of images for each filter, which is the average pooling concept. The initialization can be done using random numbers or with a constant value and train these values using backpropagation. We used trial and error method to find the initial weights and found best initial weight values as 0.1. W_{o2} is initialized using glorot normal[97]. We train these weights including the output weights W_{o2} using conjugate gradient.

Network	GAP features testing accuracy	Modified GAP features testing accuracy
CNN1 5 filters	58.19	64.7
CNN2 5 filters	63.37	66.33
CNN1 64 filters	65.95	79.19
CNN2 64 filters	70.35	73.88
CNN3 5 filters	60.45	78
CNN4 5 filters	65.9	79.47
CNN3 64 filters	72.38	77.77
CNN4 64 filters	79.25	80.6

Table 7.4: Testing accuracy for GAP and modified GAP extracted features from output discriminants from CNN

Table 7.4 shows testing accuracy for GAP and modified GAP features. We can see that modified GAP shows better accuracy than GAP. The pattern storage for modified GAP features is more than GAP feature but less than the original features.

7.3 Factorization of Output Weights

Although GAP is widely used for many deep networks[70] but from table 7.3 we see that GAP performs poorly for shallow networks. In this subsection, we modify the sparsely connected pooling network shown in figure 4.7, with a fully connected MLP with hidden unit number equal to the number of filters used in the final convolution layer and show that the output weights of the final convolution layer are factorized. We then compare the results with original results shown in subsection 4.4

Hypothesis - The output weight matrix $\mathbf{W_o}$ a CNN trained with shifted data is factorable as from figure 4.6, 4.7 and equations (4.3) through (4.8), the flattened 2-D output weight matrix $\mathbf{W_o}$ in a shallow, shift invariant CNN can be factored as

$$\mathbf{W_o} = \mathbf{W_{o2}} \cdot \mathbf{W_{o1}} \tag{7.2}$$

where $\mathbf{W_o}$ is M by N_{Fi}, $\mathbf{W}_{o1}$ is M by K and $\mathbf{W}_{o2}$ is K' by N_{Fi}. Here N_{Fi} is the number of relu features in the feature layer so N_{Fi} is $K \cdot N_{rb} \cdot N_{cb}$, where, K' is the number of filters and M is the number of classes.

Lemma - There are an uncountably infinite number of factorizations of $\mathbf{W_o}$ in equation 7.2.

Proof: The output weight matrix $\mathbf{W_o}$ in equation (7.2), can be factorized by using the flattened output weight matrix as follows.

$$\mathbf{W_{o2}} \cdot \mathbf{W_{o1}} = \mathbf{W_{o2}} \cdot \mathbf{A} \cdot \mathbf{A^{-1}} \cdot \mathbf{W_{o1}} \tag{7.3}$$

where,

$$\mathbf{W'_{o2}} = \mathbf{W_{o2}} \cdot \mathbf{A} \tag{7.4}$$

and

$$\mathbf{W'_{o1}} = \mathbf{A^{-1}} \cdot \mathbf{W_{o1}} \tag{7.5}$$

We have already shown in Table 7.3 that the linear classifier for GAP has poor performance, but good performance for no pooling. There are uncountably many non singular K by K matrices A. However, as indicated in figure. 7.5, a linear MLP can be trained using targets equal to output layer net functions as $t_p(i) = n_{po}(i)$, and iteratively adjusted to minimize the classification probability of error P_e. One advantage of factorability is that the number of features are is greatly reduced, which can make overtraining unlikely. Thus we can model **Wo** as in equation (7.2).

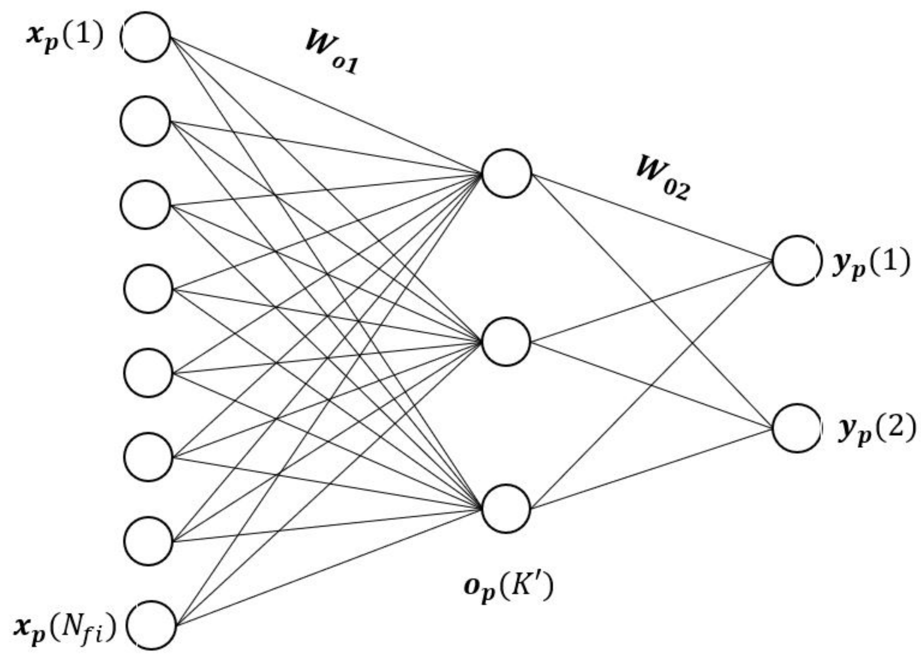


Figure 7.5: Linear MLP for M = 2, K =3

We can attempt factoriZation by training the linear MLP, where N_h is the number of filters in the CNN layer and then solve the $\mathbf{W}_{o1}$ weights iteratively using CG[64] or SD[56]. this task is performed in task 1. The second method for initializing the $\mathbf{W}_{o2}$ and $\mathbf{W}_{o1}$ weight matrices by using glorot normal[97] weight intialization and train the network using CG [64] or SD[56]. Below we demonstrate the first approach of initializing the weights using glorot normal [97] method.

Consider the shallow CNN trained in the section (4.1.2). Here, we save the feature output of the trained CNN described in section (4.1.2) as $\mathbf{x}_p$,where the size of $\mathbf{x}_p$ is N_v by N_{fi}, where N_{fi} is the number of output flattened features, we also save the output layer net functions as $t_p(i) = n_{po}(i)$. We model 2 training algorithms. The first training algorithm consists of $\mathbf{x}_p$ as inputs and output as i_c. We minimize the algorithm using linear MLP classifier with MSE- OR objective function[54][55]. The algorithm is explained in the Appendix. We named the first algorithm as LMLP-C, where the output of the final convolution layer, $\mathbf{o_{pN_L}}$ are used as inputs and output as i_c. The testing accuracy is found using best weights saved using 1-fold cross validation. Below are the results for Linear MLP classifiers for $\mathbf{o_{pN_L}}$ features. The results are given in Table 7.5

Network	K’	Testing Accuracy for factorable $\mathbf{W_o}$	Original Testing Accuracy
CNN1 5 filters	5	85.15	85.31
	200	86.15	85.31
CNN2 5 filters	5	80.83	84.20
	50	83.92	84.20
CNN1 64 filters	64	92.03	91.07
	20	92.41	91.07
CNN2 64 filters	64	92.55	89.90
	50	92.8	89.90
CNN5 64 filters	64	90.99	90.35
	200	91.01	90.35
CNN3 5 filters	5	89.1	88.26
CNN4 5 filters	5	88.15	86.48
	200	88.62	86.48
CNN3 64 filters	64	96.36	95.24
CNN4 64 filters	64	96.10	94.88
	100	96.55	94.88

Table 7.5: Results for Linear classifier on extracted features from CNN

In Table 7.5 we compare linear MLP classifier results with the original testing accuracy for each of the network. The first column *Network* is the network used to extract features, second column *Nh* is the number of hidden units in the linear MLP classifier. The hidden units for each network are equal to the filter size of the final convolution layer of the networks. Column *Linear MLP Testing Accuracy* is the testing accuracy of the linear MLP classifier for $\mathbf{o_{pN_L}}$ features for each network. Column *Original Testing Accuracy* is the original testing accuracy for each network, which is copied from Table 4.1. For each of the network, in first row we first show the testing accuracy where K' is equal to K and in the second row we show the best testing accuracy achieved for particular number of hidden units. Comparing the testing accuracies we can observe that linear MLP classifier gives us best testing accuracy.

7.4 Effects of Background change on CNN training

One possible cause of poor testing accuracy for shallow CNNS with GAP is the presence of noise in the image backgrounds. Theoretically, images with high SNR should perform better, because background noise pixels can interfere during training causing the weight changes in the direction of the noise present. In this section, we test this idea by multiplying the image backgrounds by several coefficients b, that are greater than or equal to zero. Lets define $F = S + N$, where S is the signal and N denotes background noise. The new input image is found as $F = S + b \cdot N$,where b is coefficient that changes the background noise amplitude. So, we first need to find objects inside an object and then we increase or decrease the background pixel values. This is done by finding objects inside the image using object segmentation. We, then multiply the background of each image by coefficient $b = 0., .2, .4, .8, 1., 1.2, 1.4$, and $1.6, 1.8, 2.$. We then train CNN's on each of the data and compare the results of a trained CNN with original background images.

To accomplish this task, we find objects inside each image using segmentation. To find the exact location of the object we use Canny edge detection[98] to first find objects inside the image. Scrap objects can be dusty so cannot have the same background for each object. To avoid this issue we need to use some kind of threshold measure for each image to distinguish object from the background.

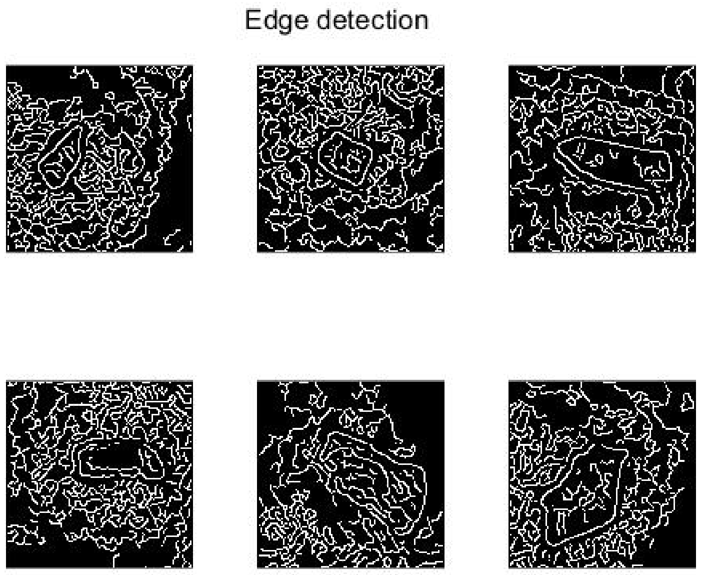


Figure 7.6: Edge detection of Random scrap images

To find thresholds for each image, we use Otsu's[99] method, Otsu's finds the thresholds based on the aggregate histogram of the entire image. We first use threshold finder for each image and use threshold with the canny edge detection. figure 7.7 displays the edge detection used with thresholding. We can also observe that some objects are not connected properly which will make us difficult to fill the object. So, we need to connect all the edges, To do that we use a circular morphological structuring element to close the image, this is performed using MATLAB's[100] *imclose function with a structuring element.* Finally, after all the edges are closed we use MATLAB's[100] *imfill* function to fill the object with all ones. We also implemented some custom code to remove extra obects in the corner of an image if present.

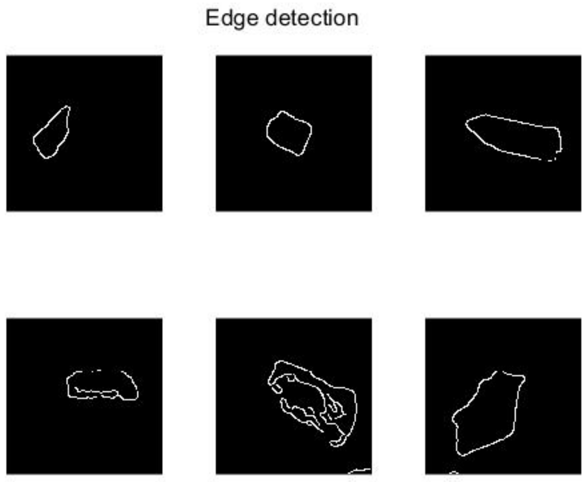


Figure 7.7: Edge detection with threshold of Random scrap images

figure 7.8 shows some scrap images from the original data. Row 1 images denote class 1 and row 2 denotes class 2.figure 7.9 shows subsequent background attenuated images of the example from figure 7.8 after segmentation.

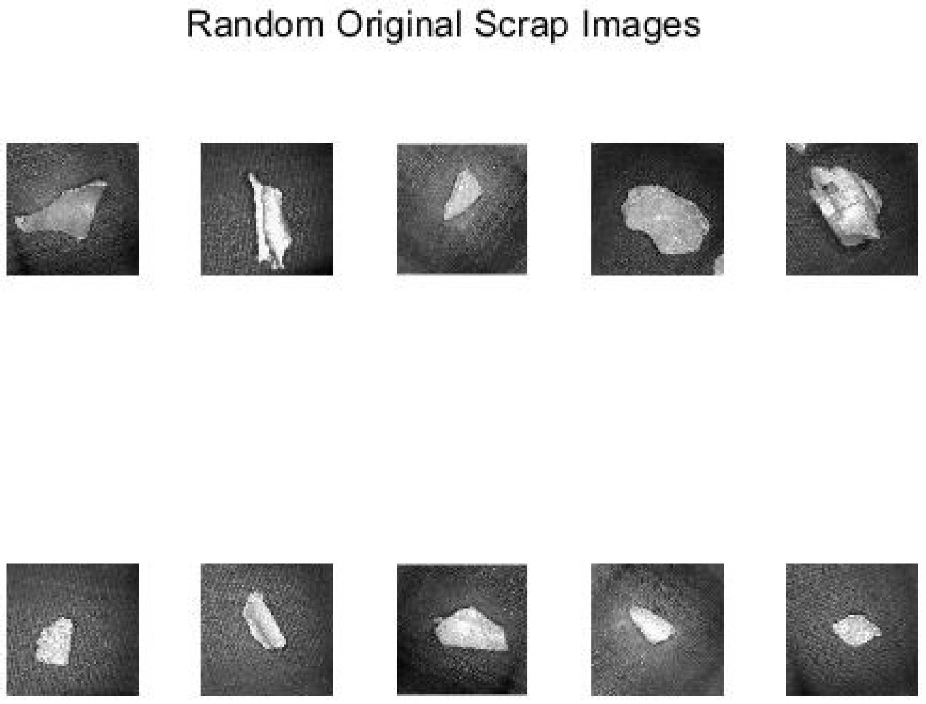


Figure 7.8: Random scrap images

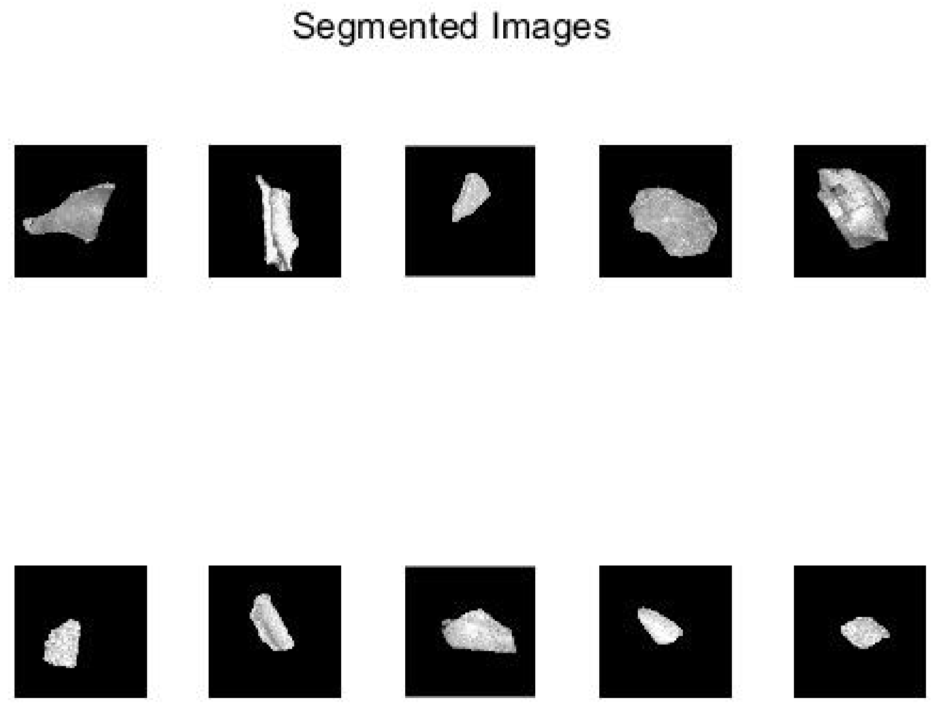


Figure 7.9: Object detected Random scrap images with zero background and 1 for objects

After cleaning the image we adjust the background by multiplying small values greater than one such as $1.2, 1.4, 1.6, 1.8, 2$ with the original background to increase background noise and multiply small values less than one but greater than or equal to zero such as $0, 0.2, 0.4, 0.6, 0.8$. figure 7.10 displays the background with original background times $0, 0.2, 0.4, 0.6, 0.8$ and original image. and figure 7.11 displays the background with original background times $1.2, 1.4, 1.6, 1.8, 2$ and original image.

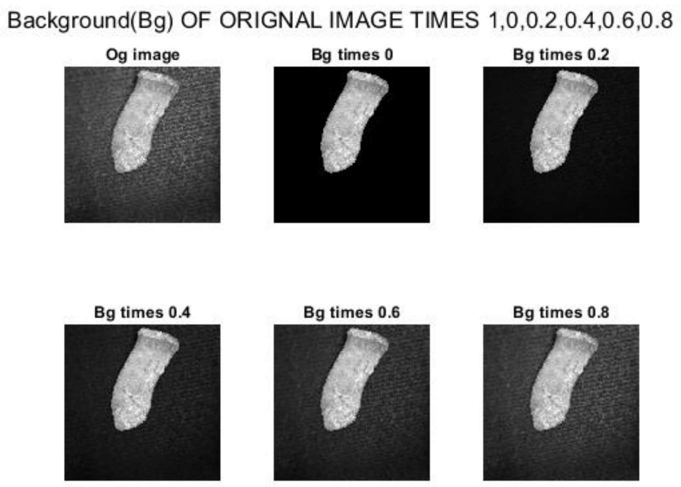


Figure 7.10: Original Background times 0 to 1 range

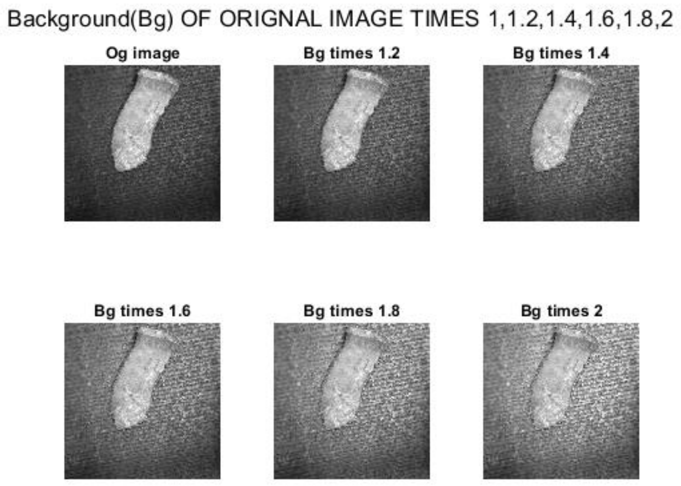


Figure 7.11: Original Background times 1 to 2 range

To find the network performance for each of the background attenuated images, we use CNN1 with 64 filters as defined in subsection 4.4. For each convolution layer used, we set the convolution strides to 1 with padding of 2 x 2 and size of filter as 3 x 3, we use adams optimizer with a initial learning rate of 0.001 and a batch size of 32. We use MATLAB's early stopping method. figure 7.12 shows testing accuracy of CNN1 with 64 filters vs scrap data versus C. The networks were initialized using glorot[97], which yeilded the best results.

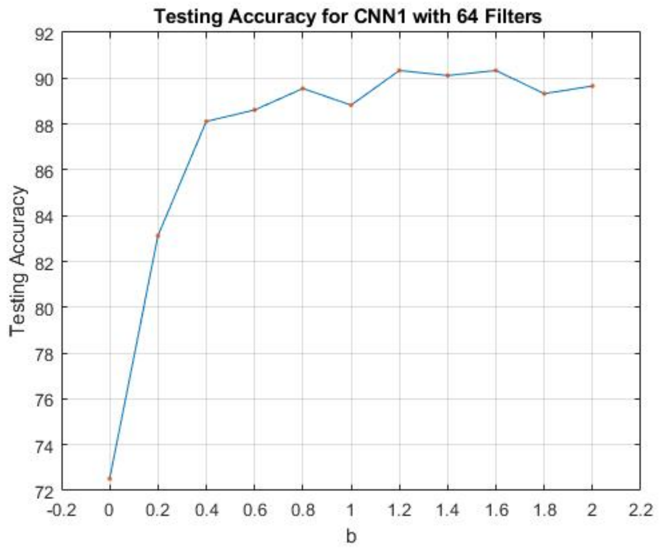


Figure 7.12: Testing accuracy for different backgrounds for CNN1 with 64 filters

From figure 7.12 we see that the original background images give us the best testing accuracy. We also observe that as the background noise increasing testing accuracy also increases. Similarly, as the background noise decreases, the testing accuracy also decreases which disproves our hypothesis that zeroing out the background increases performance.

Similarly for glorot[97], we try the above experiment with CNN2 with 64 filters figure 7.13 shows testing accuracy versus background.

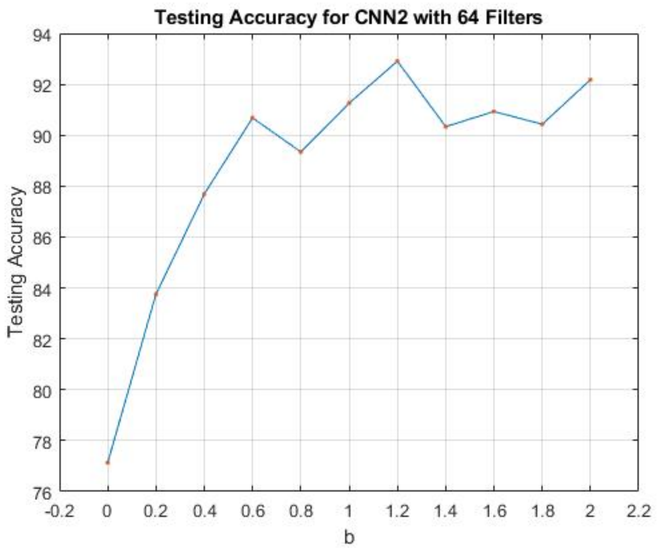


Figure 7.13: Testing accuracy for different backgrounds for CNN2 with 64 filters

From figure 7.13 we see that the original background gives us the best results but the testing accuracy for remaining coefficients is not significantly low except for background of original image multiplied by 0, 0.2 and 0.4. Also, we see that again our hypothesis of more SNR better the results is disproved.

We also compare networks for two convolution layers and a relu layer, followed by a maxpool layer with a pool size of 3x3 which we call as VGG1 layer. We show results for 2 such layers with a MLP layer for classification. The number of filters in all the convolution layer is 64 filters and the MLP with single hidden layer has 128 hidden units which can be called as VGG2. figure 7.13 shows testing accuracy of CNN2 with 64 filters vs scrap data with specific background on the x axis. We plot the results for HE[101], which were the best results.

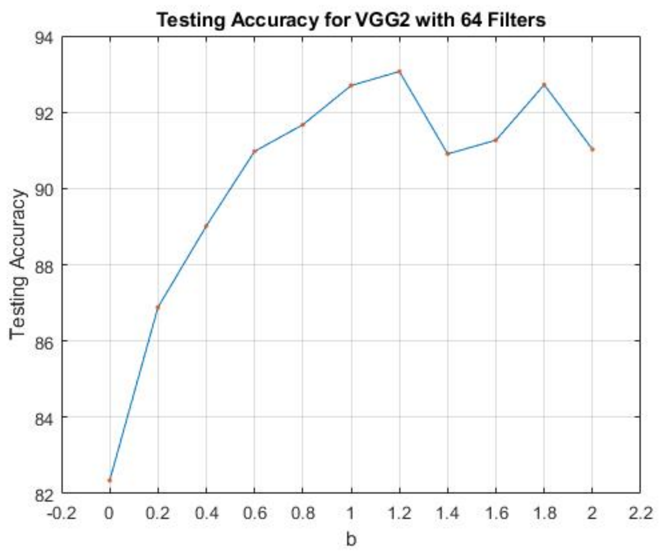


Figure 7.14: Testing accuracy for different backgrounds for CNN2 with 64 filters

From figure 7.14 we see that the background multiplied by 0.8 gives us best testing accuracy, much higher than for the original image backgrounds. Also, our hypothesis is again disproved. We also see that we get better accuracies for 1.4 ,1.6, 1.8 and 2 times the original background.

Finally, we also demonstrate a resnet18[77] structure for the background attentenuated data. The training structure is describe here[77]. We use transfer learning method where we use the learned weights from the pretrained resnet18 structure in MATLAB toolbox[100], The trained network is trained on the imagenet database[79].figure 7.15 shows testing accuracy of resnet18 structure vs scrap data with specific background on the x axis

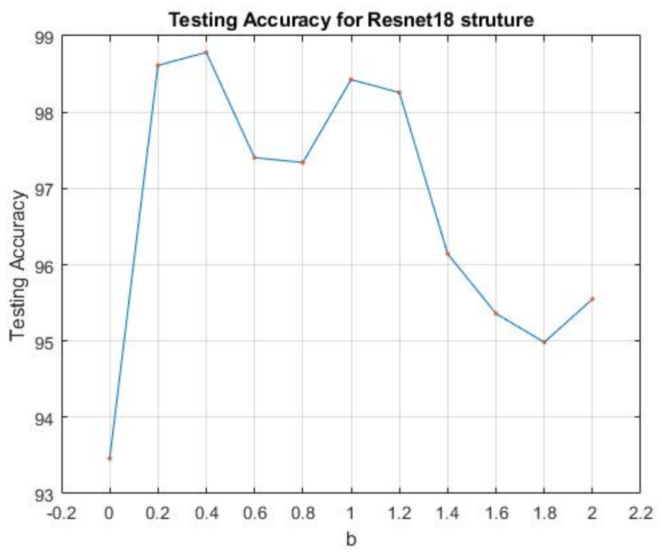


Figure 7.15: Testing accuracy for different backgrounds for Resnet18 structure

From the above figure 7.15 we can see that data with low background noise gives us the best testing accuracy. Also, we can say that a CNN training is not optimal as smaller network needs background noise and depper network need less background noise to give good testing accuracy.

Similarly, we demonstrate the segmentation experiment for coin dataset[102]. Coin dataset is used from Kaggle website. figure 7.16 shows original coin dataset and 7.17 is the segmented dataset of the original coin dataset. We have 5 different classes and about 3000 total images. No image augmentation is performed for this dataset.

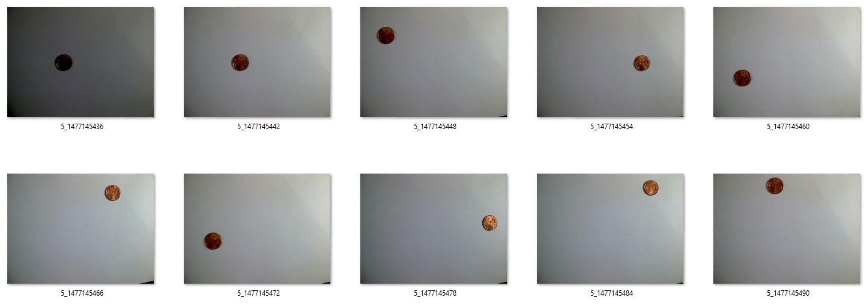


Figure 7.16: Original Coin Dataset

The segementation method used for this data is similar to the one explained for scrap datasets, except a threshold of 0.3 is used instead of using the otsu's threshold method for edge detection. This is possible because the object inside an image is a fixed round shaped. We create new dataset for each of the background coefficients used in the previous example. We then use these examples and train different CNN structures.

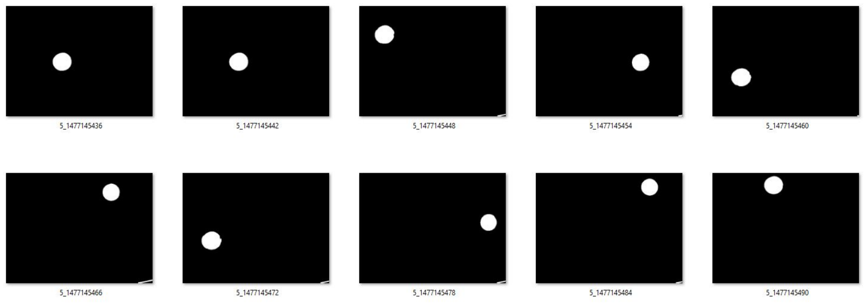


Figure 7.17: Segmented Coin Data

To find the network performance for each of the background attenuated images, we train each indivdual dataset with VGG1 and VGG2 network. From figure 7.18 and figure 7.19, we see that as the background increases so does our testing accuracy. We can also observe that we have a differnce of around 10% testing accuracy between the original backgrounds testing accuracy and 2 times the background testing accuracy.

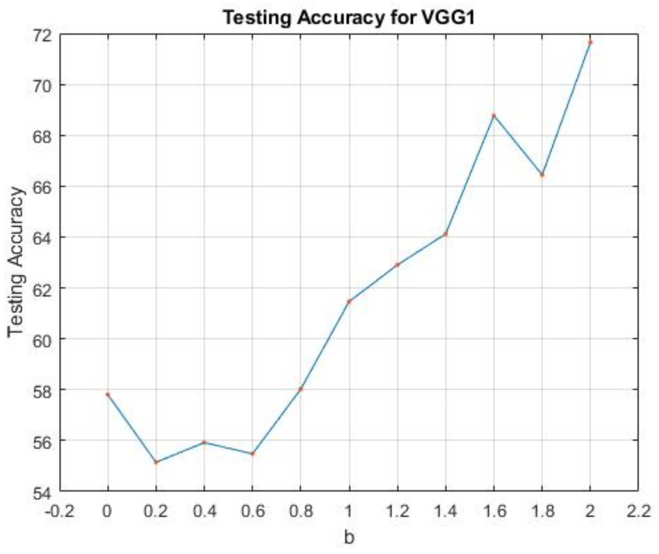


Figure 7.18: Testing accuracy for background coefficients for VGG1

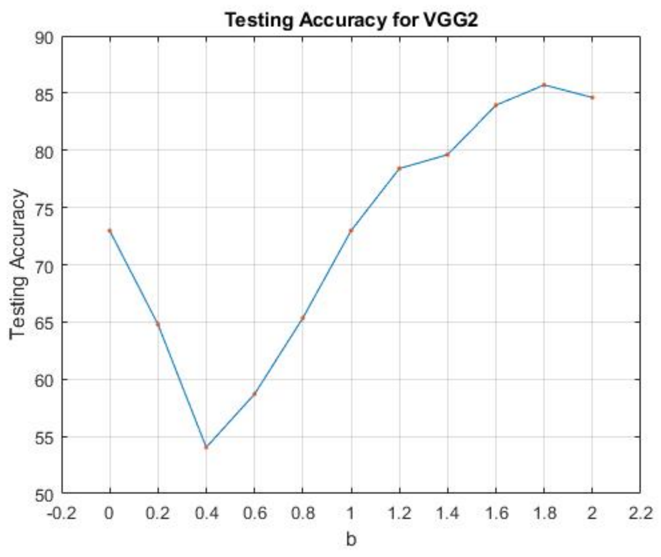


Figure 7.19: Testing accuracy for background coefficients for VGG2

From all results above we saw that increasing the background noise has shown to improve results. These results are possible as adding noise stimulates new images that prevents over training[39][103][104][105][106]. One more reason is dithering[107], which means intentionally applying noise to the inputs to prevent overfitting.

7.5 Final Thoughts on Shift Invariance

(A1) So far we have observed that global average pooling doesn't work for shallow networks. GAP-WGAP works better than GAP alone. Also, training by concatenation of global average pool features and global max pool features gives better results than global max or average pooling alone.

(A2) Factorization of the output weight matrix which is a generalization of GAP, showed good results for shallow networks as shown in section 7.3.

(A3) CNN training algorithms are not optimal as they fail to perform better for less background noise than more background noise.

(A4) Data with more background noise has been shown to work better for shallow networks than in deeper networks.

Chapter 8

PWL Adaptive Activations Work

In chapter 7, we observed that CNN training algorithms are not optimal. Ramp or relu activations are not optimal, because using relu activations for all applications does not give us best results. In this chapter, we demonstrate the PWL activations which are continuous and bounded for CNNs. These activations adapt during training and has universal approximation [44]. We compare our designed activation function with widely used relu activation for CNN's in MATLAB and PYTORCH enviornments. Finally, we also show a growing approach for selecting the best samples for a PWL activation.

8.1 Notation and Calculations of PWL Activation

PWL activations in subsection 5.3 have some limitations. The calculation fails when the distance between heights of two hinges are very close, which can happen as we train the hinges. In this section, we derive new notation and calculation of PWL activation activation using linear interpolation.

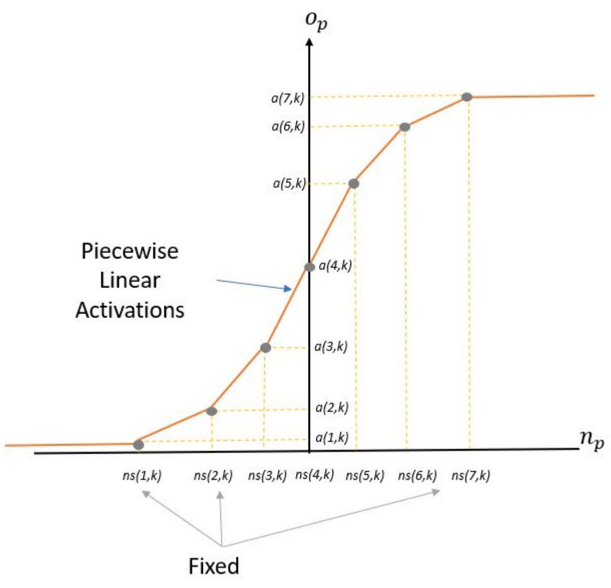


Figure 8.1: Piecewise Linear Curve

figure 8.1, show a PWL activation for K hidden units which consists of multiple ramps where, $ns(1,k)$ is the first hinge of the k^{th} hidden unit and $a(1,k)$ is its activation value. These hinge values ns are constant throughout training. From the figure we can observe that the PWL curve which passes through the activation of each of the 7 hinges. We define total number of hinges as H. The equation for above figure is given in equation 8.3. Let s denote maximum value of a net function and r denote minimum value of net function. The activations are calculated between two points, where, the net values between first two hinges are calculated with $ns(1,k)$ denoted as m_1 and $ns(2,k)$ denoted as m_2. Similarly, activations output between next two hinges are calculated by denoting $ns(2,k)$ denoted as m_1 and $ns(3,k)$ denoted as m_2. We do this for H hinges. m_1 and m_2 for each hinge is calculated as $m_1 = \lceil \frac{n_p}{\delta ns} \rceil$ and $m_2 = m_1 + 1$. Given the net function $n_p(k)$, $o_p(k)$ is calculated as,

$$w_{1p}(k) = \frac{ns(m_2,k) - n_p(k)}{ns(m_2,k) - ns(m_1,k)} \tag{8.1}$$

$$w_{2p}(k) = \frac{n_p(k) - ns(m_1,k)}{ns(m_2,k) - ns(m_1,k)} \tag{8.2}$$

$$o_p(k) = \left\{ \begin{matrix} a(H,k) & for \quad n_p(k) > s \\ w_{1p}(k) \cdot a(m_1,k) + w_{2p}(k) \cdot a(m_2,k) & for \quad s > n_p(k) > r \\ a(1,k) & for \quad n_p(k) < r \end{matrix} \right\} \tag{8.3}$$

where, $w_{1p}(k)$ and $w_{2p}(k)$ are the slope equation for each of the two hinges for k^{th} hidden unit. $n_p(k)$ is the p^{th} pattern and k^{th} hidden unit net function and $o_p(k)$ is its activation function. For all the activation values less $ns(1, k)$ has zero slope hence $n_p(k) = a(1, k)$. Similarly, for all the values greater than $ns(H, k)$ has zero slope therefore, $n_p(k) = a(H, k)$.

Example of PWL activation calculation For initialization of PWL activation, we first need to initialize the PWL using the most widely used activations such as sigmoid[40], relu[32] and leaky relu[33], etc. We then decide total number H on the net function. This method can be done individually for each of the hidden units. In this dissertation, we use same number of ns hinges for each of the hidden units. We then find the minimum and maximum hinge values from the net function output. This is achieved by randomly selecting data from each of the classes and performing convolution and then selecting the minimum and maximum value from the output of the convolution.

Below, we show the calculation for PWL activations for $k = 1$ hidden unit as follows. As discussed above we first need to find the initial activation. For this example we use sigmoid activation as shown in figure 8.2

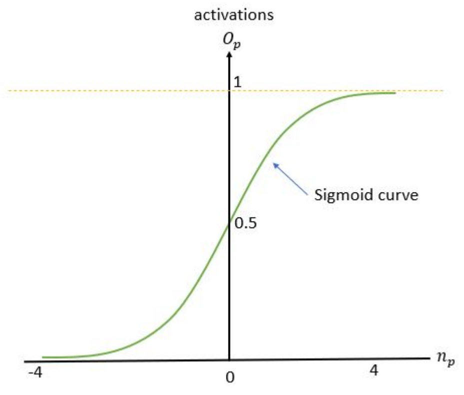


Figure 8.2: Sigmoid Curve

From the figure we can see that x axis is the net function n_p and the y axis is its corresponding sigmoid activation. The range of sigmoid is from 0 to 1. Step 2 is finding the minimum and maximum value from convolution output. In this case, we select the values as -4 and 4. Now, we need to decide ns samples on the sigmoid curve. This step is user choosen, In section 8.4 we discuss a growing and a pruning approach to find the best hinges during training. Here, for this example we choose $H = 7$. We show our selection of hinges and activations in table format as show in Table 8.1

H	1	2	3	4	5	6	7
Fixed hinges (ns_1)	-4	-2.67	-1.33	0	1.33	2.67	4
Activations for hinges(a_1)	0.02	0.07	0.21	0.5	0.79	0.94	0.98

Table 8.1: PWL samples and activations for one hidden unit

From table 8.1, we can see that there are total $H = 7$ hinges ranging from -4 to 4, which are our minimum and maximum values from the net function and its sigmoid activations as activation samples a. Finally we plot these points on the sigmoid curve from figure 8.2. The curve after plotting these points should look like figure 8.3

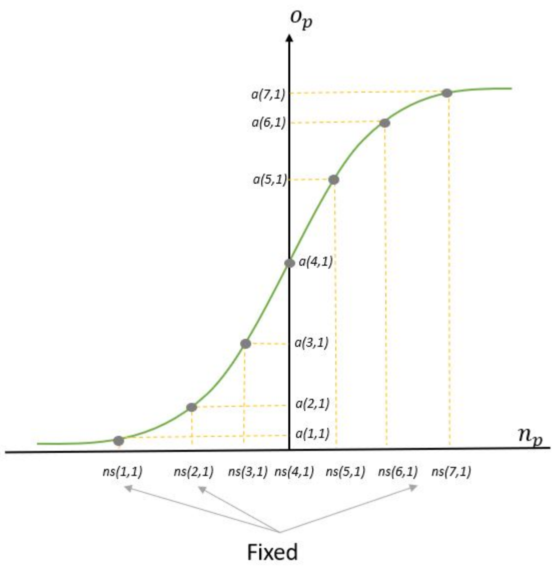


Figure 8.3: Sigmoid with Fixed Samples

figure 8.3 is the plot for a fixed piecewise sigmoid activation for net versus activations values where 7 hinges are plotted on to the sigmoid curve. Now, for the final piecewise linear curve we remove the sigmoid curve and linearly join 2 points using linear interpolation technique.

Linear interpolation involves estimating a new value of a function between two known fixed points [108].

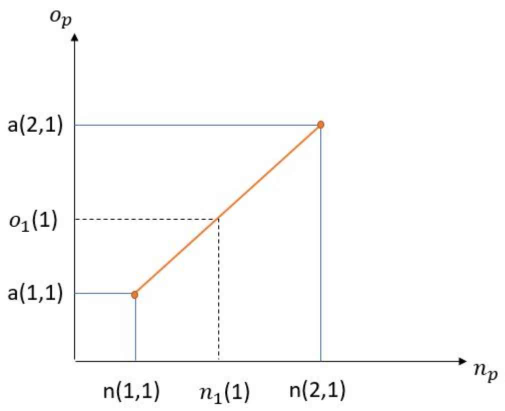


Figure 8.4: Linear interpolation between 2 points

figure 8.4 relates the use of linear interpolation between 2 fixed ns points. We can say that if we have a new sample net value $n_1(1)$, its corresponding activation value is as shown in the figure. To find $o_1(1)$ between $a(1,1)$ and $a(2,1)$, we use the following equation.

$$o_1(1) = \frac{ns(2,1) - n_1(1,1)}{ns(2,1) - ns(1,1)} \cdot a(1,1) + \frac{n_1(1) - ns(1,1)}{ns(2,1) - ns(1,1)} \cdot a(2,1) \tag{8.4}$$

Finally, we use the equation 8.3 to find all the activation output, The plot of net versus activation is shown in figure 8.1.

8.2 PWL activation training using steepest descent

Above discussed PWL activations $\mathbf{A}$ are trained via steepest descent. The negative gradient matrix $\mathbf{G_a}$ with respect to E_{ce} is calculated as,

$$g_a(k,m) = -\frac{\partial E_{ce}}{\partial a(k,m)} \tag{8.5}$$

where k is the hidden unit number and m is the H^{th} hinge.

$$g_a(k,m) = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}(t_p(i) - y_p(i)) \cdot \frac{\partial y_p(i)}{\partial a(u,m)} \tag{8.6}$$

$$\frac{\partial y_p(i)}{\partial a(u,m)} = w_{oh}(i,u) \cdot \frac{\partial o_p(i)}{\partial a(u,m)} \tag{8.7}$$

$$\frac{\partial o_p(i)}{\partial a(u,m)} = w_{oh}(i,u) \cdot ((\delta(m - m_1) \cdot w_1(p,u)) + (\delta(m - m_2) \cdot w_2(p,u))) \tag{8.8}$$

where, for the p^{th} pattern and k^{th} hidden unit of the net value we find m_1 and m_2, where the p^{th} pattern of k^{th} hidden unit of the net value lies between the two fixed piecewise linear sample values m_1 and m_2 of the u^{th} hidden unit as described in the search algorithm. Also we need to find $w_1(p,u)$ and $w_2(p,u)$ from equations 8.1 and 8.2. A search algorithm is used to find the correct m sample for a particular pattern's hidden unit is found[94]. The equation 8.8 solves for the p^{th} patterns u^{th} hidden unit of the piecewise linear activations and accumulates the gradient for all the p^{th} patterns of their respective u^{th} hidden units.

Adams optimizer[67] is used to find learning factor and update the weights of the activation. which are updated as follows

$$\mathbf{A} = \mathbf{A} + z \cdot \mathbf{G_a} \tag{8.9}$$

8.3 Structure and Results Using Adaptive Activation

In this subsection, we first describe the PWL initialization and structure used for training the CNN with adaptive activation. We then compare results for CNN with adaptive activation and RELU for MATLAB and PYTORCH enviornments.

8.3.1 PWL Initialization

In this section, we describe the PWL initialization and structure for PWL algorithm. Algorithm 1 is used for both the MATLAB and PYTORCH structure. This algorithm involves initialization of the PWL hinges and finding the minimum and maximum values for hinges in each layer.

Algorithm 1 Pseudo-code for PWL initialization algorithm

```
1: Select the CNN structure which includes convolution and MLP layers indicated as $N_L$.
2: Select total number of samples($H$) in each layers.
3: Initialize weights for the first convolution layer and calculate $net_1$
4: Find minimum $ns(1)$ and maximum $ns(H)$ from $net_1$
5: Using these hinges find $H - 2$ equidistant hinges between $ns(1)$ and $ns(H)$
6: Activation values $a$ for these $H$ $ns$ samples are user choosen. But Linear activation is prefered
   if MSE-OR objective functions are used.
7: Similarly, Do step 3 to 6 for $N_L$ layers
8: Run the above code for $N_L$ x 2.
```

The forward propagation for the PWL-CNN is similar to the structure with relu activation function. Here, we replace the relu layer with PWL activations. And during backpropogations gradients are calculated using equation 8.5. And finally the activations are updated using equation 8.9.

8.3.2 Testing comparison for MATLAB Structure

In this subsection, we will compare our PWL-CNNs versus RELU-CNNs. The CNN with adaptive activation code is written from scratch in MATLAB. We compare two different structures. Structure one uses a softmax cross entropy objective function and structure two use MSE - Output reset(OR)[54][55] objective function. Structure one will be used for MATLAB's inbuilt code with relu activations and structure two will be used for CNN with PWL adaptive activations described in appendix. We first use a basic structure for comparison, which includes one convolution layer with a linear layer for each of the two structures.

Here we compare the results for cross-entropy with softmax and relu activations (SCE-RELU) algorithm and mean square error with output reset and PWL activations(MSEOR-PWL) algorithm and show that PWL activations perform better than relu activations in most cases. The results for SCE-RELU are generated using MATLABs deep learning toolbox[100]. The structure of the algorithm is described in the appendix. Weight initializer method used softmax cross-entropy algorithm is glorot normal[97]. Input images are normalized between 0 and 1 . The data is shuffled at every iteration. The filter size used is 3 x 3 and $L = 1$ convolution layers. Training is performed using Adams optimizer with a learning rate as 0.001 and default parameters in [67]. Below table shows results for 5 and 20 filters respectively.

Data	SCE-RELU testing accuracy	MSEOR-PWL testing accuracy
MNIST	97.47	98.05
cifar10	51.57	57.27
cifar100	21.64	28.54
Scrap	85.56	91.52
svhn	78.95	80.47
fashion mnist	88.62	89.72
Intel image	64.37	69.9

Table 8.2: SCE-RELU vs MSEOR-PWL for 5 filters and $N_L = 1$ convolution layer

In Table 8.2 and Table 8.3 we compare the MSEOR-PWL and SCE-RELU algorithm structures using 5 and 20 filters respectively. The network with the best validation accuracy is used to retrain the network with both training and validation data merged to get the best testing accuracy. The number of training iterations used is that which produces best validation accuracy. The best testing accuracy rows are colored. We see that testing results for MSEOR with pwl adaptive activations gives the best testing accuracy for almost the datasets.

Data	SCE-RELU testing accuracy	MSEOR-PWL testing accuracy
MNIST	97.85	98.18
cifar10	59.063	64.73
cifar100	29.22	32.77
Scrap	89.43	92.1
svhn	80.83	80.78
fashion mnist	89.53	90.61
Intel image	69.27	71.3

Table 8.3: SCE-RELU vs MSEOR-PWL for 20 filters and $N_L = 1$ convolution layer

Similarly, we use a bigger network that consists of two convolution layers and a single linear classification layer.

Data	SCE-RELU testing accuracy	MSEOR-PWL testing accuracy
MNIST	98.53	98.6
cifar10	61.69	66.84
cifar100	30.67	38.34
Scrap	94.97	94.55
svhn	83.84	86.02
fashion mnist	90.21	90.32
Intel image	70.93	74.6

Table 8.4: SCE-RELU vs MSEOR-PWL for 20 filters and $N_L = 2$ convolution layer

From Table 8.7, shows results for SCE-RELU vs MSEOR-PWL for 20 filters and 2 convolution layers. Here we observe that, MSEOR-PWL outperforms for 3-dimension colored input datasets such as cifar10, cifar100, svhn and intel image. In next table we show how many layer or filters a SCE-RELU CNN needs to get results similar to MSEOR-PWL for cifar10, cifar100, svhn and intel image datasets.

	SCE-RELU			MSEOR-PWL		
Data	layers	filters between layers	testing accuracy	layers	filters between layers	testing accuracy
cifar10	C-C-P-F	32-64-32	67.17	C-C-L	20-20	66.84
cifar100	C-C-P-F	32-64-32	32.5	C-C-L	20-20	38.34
svhn	C-C-P-F	32-32-32	85.82	C-C-L	20-20	86.02
Intel image	C-C-C-P-F	64-64-128-128	74.37	C-C-L	20-20	74.6

Table 8.5: Number of filters or layers needed for relu to achieve similar results

where column *layer* indicates number of convolution, max pooling and fully connected layers are used. Here, C defines convolution plus relu layer, P defines maxpooling for size 2x2 with stride 1, F defines an MLP classfication layer and L defines a linear classification layer. *filters between layers* indicates the number of filters in convolution layer and hidden units in MLP layer. We can see that all the networks need an MLP and a maxpool layer to achieve the same results. More filters were tried before adding extra layers, but the results were still worse than for MSEOR-PWL.

8.3.3 Testing comparison for PYTORCH Structure

In this subsection, we compare our CNN with PWL adaptive activations and CNN with relu activation in Pytorch. We also simultaneously compare the results for softmax cross-entropy with relu activations (SCE-RELU) algorithm and softmax cross-entropy with PWL activations(SCE-PWL). The weight initializer method used for SCE-RELU is HE normal[101]. Input images are normalized between 0 and 1 and the data is shuffled at every iteration. The filter size used is 3 x 3 and training is performed using Adams optimizer with a learning rate as 0.001 and default parameters in [67]

Data	SCE-RELU testing accuracy	SCE-PWL testing accuracy
MNIST	98.88	99.04
cifar10	66.42	67.54
cifar100	25.58	29.41
Scrap	93.44	91.21
svhn	84.24	85.58
fashion mnist	91.45	92.31
Intel image	75.73	76.77

Table 8.6: SCE-RELU vs SCE-PWL for VGG1 structure

Table 8.6 shows the results for VGG1 structure layer, here VGG1 structure layer mean two convolution layers with 32 filters followed by a maxpool layer and single hidden layer ML with 64 hidden units. From the table we observe that SCE-PWL works better for most of the cases but the results using softmax cross entropy objective function doesn't give significant results as using mean square error and output reset. In next table we show how many layer or filters a SCE-RELU CNN needs to get results similar to MSE-OR for MNIST, cifar10, cifar100, svh, nfashion mnist and intel image datasets.

	SCE-RELU			MSE-PWL		
Data	layers	filters between layers	testing accuracy	layers	filters between layers	testing accuracy
mnist	C-C-P-F	64-64-128	99.01	C-C-P-F	32-32-64	99.04
cifar10	C-C-P-F	64-64-128	66.77	C-C-P-F	32-32-64	67.54
svhn	C-C-P-F	64-64-128	84.15	C-C-P-F	32-32-64	85.58
svhn	C-C-P-F	64-64-128	84.9	C-C-P-F	32-32-64	85.58
fashion mnist	C-C-P-F	64-64-128	92.150	C-C-P-F	32-32-64	92.31
Intel image	C-C-P-F	64-64-128	76	C-C-P-F	32-32-64	76.77

Table 8.7: SCE-RELU vs MSE-PWL layer comparison

where column $layer$ indicates number of convolution, max pooling and fully connected layers are used. Here, C defines convolution plus relu layer, P defines maxpooling for size 2x2 with stride 1, and F defines fully connected MLP layer. $filtersbetweenlayers$ indicates the number of filters in convolution layer and hidden units in MLP layer. We can see that all the networks need an additional MLP layer and maxpooling to achieve the same results. More filters were tried before adding extra layers, but the results were still worse than MSE-PWL and that is because of the objective function and the training enviornment.

Data	SCE-RELU testing accuracy	SCE-PWL testing accuracy
MNIST	99.1	99.41
cifar10	70.92	72.56
cifar100	26.76	34.02
Scrap	95.72	97
svhn	87.38	90.24
fashion mnist	92.17	92.39
Intel image	78.2	79.9

Table 8.8: SCE-RELU vs SCE-PWL for VGG2 structure

Table 8.8 discusses the results for VGG2 layer, here VGG2 layer is two VGG1 layer, where VGG1 layer is two convolution layers followed by a maxpool layer, with a single hidden layer MLP an objective function. The first two convolution layers has 32 filters the next two convolution layers has 64 filter and the hidden units in the MLP layer is 128 .From the table we observe that SCE-PWL works better for all of the cases but the results using softmax cross entropy objective function don't give significantly better results than using mean square error and output reset.

8.4 Growing and Pruning for PWL Hinges

In this subsection, we discuss a growing and a pruning approach to find the best ns hinge values. We show the best number of hinge values for each datasets.

There are many ways we can grow and prune hinge values. One way to grow and prune the values can be by first intializing the PWL hinge with a small value such value of H as 3 and then grow at each iteration between 2 hinges only if validation accuracy increases. If the validation accuracy does not increase, we can prune each of the hinge value and again check if the validation accuracy increases. This is an iterative process but this can lead to network getting stuck at small hinge values. To avoid this problem we grow and simulataneously prune the hinges after each activation. Below we show the best approach to grow and pune the hinge values

In this approach, we observed that in previous section that H = 9 gave good testing accuracy, so we can start by intializing the PWL hinges with H = 9. After the first iteration, we find the best

PWL samples by growing and prunning after each iteration. We grow and prune based on the best validation accuracy. After every iteration, we first add a sample point between each pair of samples between $ns(1)$ and $ns(H)$. The activation value of each of the hinge is equal to the ns value, which in this case is linear. If the best validation accuracy is found we grow the samples. Simultaneously, we prune each samples if we get the best validation accuracy between $ns(1)$ and $ns(samples)$. The pseudo code is provided in table 2

Algorithm 2 PWL Growing algorithm

```
1: Divide training data into training and validation data
2: Normalization of training, validation and testing data
3: Initialize initial PWL acivations with H = 9, initialize filters and output weights.
4: for i=0,i<$N_{it}$, i++ do
5:     Perform forward propagation
6:     Calculate all gradients through back propagation and update all weights.
7:     Calculate validation accuracy and save weights for best accuracy and iterations.
8:     Grow samples between 2 hinges(H) and calculate validation accuracy for each added H.
9:     If validation accuracy increases update weights(add samples) and continue training.
10:    Prune samples one by one and calculate validation accuracy for each pruned hinge
11:    If validation accuracy increases update weights(add samples) and continue training
12:    Save weights for best validation accuracy and simulataneously save best iteration $best_it$
13: end for
14: Merge Training and validation data and train with the best parameters for $best_it$ iterations.
```

Data	Optimized H	MSE-PWL Testing accuracy	Constant H	Previous MSE-PWL Testing accuracy
MNIST	8	98.10	9	98.18
cifar10	10	64.38	9	64.73
cifar100	7	32.33	9	32.77
Scrap	13	93.66/94.55	9	92.1
svhn	8	80.27	9	80.78
fashion mnist	11	91.04	9	90.1
Intel image	8	71.57	9	71.3

Table 8.9: Growing results for one CNN layer with adaptive activation and a linear classifier

Table 8.9 shows growing and a pruning results for one convolutional layer with 20 filters and a linear classifier. We compare these results with that of Table 8.3. Comparing Table 8.9 and 8.3, we observe that except for the svhn dataset, our designed MSE-PWL CNN achieves better results than SCE-RELU CNN. Also, we see improved results for scrap, fashion mnist and intel image and mnist compared to the MSE-PWL with 9 fixed hinges. We can also observe that some datasets need more hinges and some need less hinges to achieve similar or better results.

Chapter 9

Conclusions

In this dissertation, we first review traditional methods for introducing shift invariance in CNNs. We find that traditional global pooling approaches don't work well for shallow CNNs. We find that regular shallow CNNs that are trained on shifted images have factorable output weight matrices, as in GAP. The difference is that the W_{o2} matrix is sparse for GAP. This is done by inserting the final features into a linear MLP, which is then trained using actual target classes. A simple segmentation example is also shown where we change the background noise by increasing and also decreasing the background pixel intensities. Here we see that for the scrap data example, CNNs failed to perform better for zero background.

A new adaptive piecewise linear activation has been introduced. We show results in MATLAB and Pytorch environments. Results show that CNN with PWL activation works better than relu activations. With inclusion of new adaptive activation, the function involves multiple hyperparameter selections. In this dissertation, we solve most of the hyper parameter problems by using initial activations as linear activation for MSE-PWL in MATLAB, and leaky relu activation for SCE-PWL in PYTORCH. The number of piecewise linear samples are selected using a growing and pruning approach to find the best piecewise linear samples. We also saw that PWL gives us better results with MSE-OR objective function with comparatively less number of filters and layers than SCE objective functions. Applications with a curved output involve using more number of PWL samples to approximate.

Chapter 10

Future Work

In this dissertation, we observed that in section 7.3 the output weights can be factored for CNN with shifted data. Future work can involve investigating how often CNN output weight factorization helps in performance. Similarly, a more efficient PWL activation in deep CNN can be investigated, where whole network weights including PWL hinge intialization can be improved. In this dissertation, we train the activations of the hinge values. Training of hinge locations can also be tried separately or along with the activations. Growing and pruning of the hinges can be extended and implemented as a pre-processing step.

Bibliography

[1] D. Rumelhart, Geoffrey E. Hinton, and R. J. Williams. Learning internal representations by error propagation. 1986.

[2] Michael I. Jordan. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495, 1997.

[3] J J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[5] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[7] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[8] Wei Zhang. Shift-invariant pattern recognition neural network and its optical architecture. *Proceedings of Annual Conference of the Japan Society of Applied Physics*, 1988.

[9] Wei Zhang, Kouichi Itoh, Jun Tanida, and Yoshiki Ichioka. Parallel distributed processing model with local space-invariant interconnections and its optical architecture. *Applied optics*, 29 32:4790–7, 1990.

[10] Jeff Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389, 2014.

[11] Yun Liu, Guolei Sun, Yu Qiu, Le Zhang, Ajad Chhatkuli, and Luc Van Gool. Transformer in convolutional neural networks. *CoRR*, abs/2106.03180, 2021.

[12] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. *CoRR*, abs/1506.04214, 2015.

[13] Varun Gulshan, Lily Peng, Marc Coram, Martin C. Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip C. Nelson, Jessica L. Mega, and Dale R. Webster. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA*, 316(22):2402–2410, 12 2016.

[14] Paras Lakhani and Baskaran Sundaram. Deep learning at chest radiography: Automated classification of pulmonary tuberculosis by using convolutional neural networks. *Radiology*, 284(2):574–582, 2017. PMID: 28436741.

[15] Richard Kijowski, Fang Liu, Francesco Caliva, and Valentina Pedoia. Deep learning for lesion detection, progression, and prediction of musculoskeletal disease. *Journal of Magnetic Resonance Imaging*, n/a(n/a).

[16] Kuprel B. Novoa R Esteva, A. Dermatologist-level classification of skin cancer with deep neural networks. In *Nature 542*, page 115–118, 2017.

[17] Hardik Nahata and Satya P. Singh. *Deep Learning Solutions for Skin Cancer Detection and Diagnosis*, pages 159–182. Springer International Publishing, Cham, 2020.

[18] Earnest Paul Ijjina and Krishna Mohan Chalavadi. Human action recognition using genetic algorithms and convolutional neural networks. *Pattern Recognition*, 59:199 – 212, 2016. Compositional Models and Structured Learning for Visual Recognition.

[19] German I. Parisi. Human action recognition and assessment via deep neural network self-organization, 2020.

[20] Patrik Kamencay, Miroslav Benco, Tomas Mizdos, and Roman Radil. A new method for face recognition using convolutional neural network. *Advances in Electrical and Electronic Engineering*, 15, 11 2017.

[21] Antonio J. Colmenarez and Thomas S. Huang. *Face Detection and Recognition*, pages 174–185. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[22] Patrice Simard, David Steinkraus, and John Platt. Best practices for convolutional neural networks applied to visual document analysis. pages 958–962, 01 2003.

[23] S. Marinai, M. Gori, and G. Soda. Artificial neural networks for document analysis and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:23–35, 2005.

[24] Rumelhart Hinton and R. J. Williams. Learning internal representations by error propagation. 1:318–362, 1986.

[25] W. Kaminski and P. Strumillo. Kernel orthonormalization in radial basis function neural networks. *IEEE Transactions on Neural Networks*, 8(5):1177–1183, Sep. 1997.

[26] R P Morgan Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4:4–22, 1987.

[27] Pramod Lakshmi Narasimha, Walter Delashmit, Michael Manry, Jiang Li, Francisco Maldonado, and Dr Zhang. An integrated growing-pruning method for feedforward network training. *Neurocomputing*, 71:2831–2847, 08 2008.

[28] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–435, 1952.

[29] Jonathan R Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, USA, 1994.

[30] Christakis Charalambous. A conjugate gradient algorithm for the efficient training of artificial neural networks. Technical report, 1990.

[31] Roger Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, New York, NY, USA, second edition, 1987.

[32] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. volume 27, pages 807–814, 06 2010.

[33] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[34] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 11 2018.

[35] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *CoRR*, abs/1805.12177, 2018.

[36] Toshio Akabane Yoshiji Fujimoto Kouichi Yamaguchi, Kenji Sakamoto. A neural network for speaker-independent isolated word recognition. *First International Conference on Spoken Language Processing (ICSLP 90)*, 1990.

[37] Sparsh Mittal. A survey of fpga-based accelerators for convolutional neural networks. *Neural Computing and Applications*, 32:1109–1139, 2018.

[38] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. JMLR Workshop and Conference Proceedings.

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[40] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[41] Kamel Abdelouahab, Maxime Pelcat, and Francois Berry. Why tanh is a hardware friendly activation function for cnns. In *Proceedings of the 11th International Conference on Distributed Smart Cameras*, ICDSC 2017, page 199–201, New York, NY, USA, 2017. Association for Computing Machinery.

[42] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998.

[43] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[44] Andrei Nicolae. PLU: the piecewise linear unit activation function. *CoRR*, abs/1809.09534, 2018.

[45] Samantha Guarnieri, Francesco Piazza, and Aurelio Uncini. Multilayer feedforward networks with adaptive spline activation function. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 10:672–83, 02 1999.

[46] Phillip J. Barry and Ronald N. Goldman. A recursive evaluation algorithm for a class of catmull-rom splines. SIGGRAPH '88, page 199–204, New York, NY, USA, 1988. Association for Computing Machinery.

[47] P. Campolucci, F. Capperelli, S. Guarnieri, F. Piazza, and A. Uncini. Neural networks with adaptive spline activation function. In *Proceedings of 8th Mediterranean Electrotechnical Conference on Industrial Applications in Power Systems, Computer Science and Telecommunications (MELECON 96)*, volume 3, pages 1442–1445 vol.3, 1996.

[48] Ameya Dilip Jagtap, K. Kawaguchi, and G. Karniadakis. Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks. *Proceedings of the Royal Society A*, 476, 2020.

[49] Forest Agostinelli, M. Hoffman, Peter Sadowski, and P. Baldi. Learning activation functions to improve deep neural networks. *CoRR*, abs/1412.6830, 2015.

[50] Alex Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[51] P. Baldi, P. Sadowski, and D. Whiteson. Enhanced higgs boson to+search with deep learning. *Physical Review Letters*, 114(11), Mar 2015.

[52] Zhilu Zhang and Mert R. Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8792–8802, Red Hook, NY, USA, 2018. Curran Associates Inc.

[53] Claude Sammut and Geoffrey I. Webb, editors. *Mean Squared Error*, pages 653–653. Springer US, Boston, MA, 2010.

[54] R. G. GORE, J. LI, M. T. MANRY, L. M. LIU, C. YU, and J. WEI. Iterative design of neural network classifiers through regression. *International Journal on Artificial Intelligence Tools*, 14(01n02):281–301, 2005.

[55] Jiang Li, Michael Manry, Li-min Liu, Changhua Yu, and John Wei. Iterative improvement of neural classifiers. volume 2, 01 2004.

[56] C Lemaréchal. Cauchy and the gradient method. *doc Math extra*, pages 251–254, 2012.

[57] C Lemaréchal. The method of steepest descent for non-linear minimization problems. *Quart. Appl. Math. 2*, pages 258–261, 1944.

[58] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, page 9–50, Berlin, Heidelberg, 1998. Springer-Verlag.

[59] Stiefel Magnus Rudolph, Hestenes; Eduard L. *Method of conjugate gradients for solving linear systems,*. National Bureau of Standards, 1952.

[60] Gene H. Golub and Charles F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, third edition, 2012.

[61] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. http://www.deeplearningbook.org.

[62] Dennis Ruck, Steven Rogers, Matthew Kabrisky, Mark Oxley, and B. Suter. The multilayer perceptron as an approximation to a bayes optimal discriminant function. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 1:296–8, 02 1990.

[63] M. T. Manry. Statistical pattern recognition. Summer, 1994.

[64] J. P. Fitch, S. K. Lehman, F. U. Dowla, S. Y. Lu, E. M. Johansson, and D. M. Goodman. Ship wake-detection procedure using conjugate gradient trained artificial neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 29(5):718–726, Sep. 1991.

[65] Siddharth Mahendran, Haider Ali, and René Vidal. 3d pose regression using convolutional neural networks. *CoRR*, abs/1708.05628, 2017.

[66] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Image orientation estimation with convolutional networks. volume 9358, pages 368–378, 10 2015.

[67] Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[68] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[69] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[70] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. 12 2013.

[71] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[72] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[73] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations?, 2019.

[74] Marco Manfredi and Yu Wang. Shift equivariance in object detection, 2020.

[75] Richard Zhang. Making convolutional networks shift-invariant again. *CoRR*, abs/1904.11486, 2019.

[76] Gao Huang, Zhuang Liu, and K. Weinberger. Densely connected convolutional networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017.

[77] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[78] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016.

[79] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[80] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017.

[81] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[82] Geoffrey Hinton. Pooling layers in convolutional neural networks, 2015. [Online; accessed 6-September-2014].

[83] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3856–3866. Curran Associates, Inc., 2017.

[84] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with EM routing. In *International Conference on Learning Representations*, 2018.

[85] K. V. D. Sande, J. Uijlings, T. Gevers, and A. Smeulders. Segmentation as selective search for object recognition. *2011 International Conference on Computer Vision*, pages 1879–1886, 2011.

[86] Jasper Uijlings, K. Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 09 2013.

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[88] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[89] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[90] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[91] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[92] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[93] Kamel Abdelouahab, M. Pelcat, and François Berry. Why tanh can be a hardware friendly activation function for cnns. pages 199–201, 09 2017.

[94] Chinmay Appa Rane. Multilayer perceptron with adaptive activation function. *Masters Thesis*, 2016.

[95] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[96] Lu Lu, Yeonjong Shin, Yanhui Su, and George Karniadakis. Dying relu and initialization: Theory and numerical examples, 03 2019.

[97] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[98] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[99] N. Otsu. A threshold selection method from gray level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9:62–66, 1979.

[100] Inc. The MathWorks. *Symbolic Math Toolbox*. Natick, Massachusetts, United State, 2019.

[101] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 1026–1034, USA, 2015. IEEE Computer Society.

[102] Luis Moneda; David Yonekura; Elloá Guedes. Brazilian coin detection dataset, 2020.

[103] Russell D. Reed and RobertJ Marks II. *Neural Smithing.* Bradford, USA, 1999.

[104] Christopher M Bishop. *Neural Networks for Pattern Recognition.* USA, 1995.

[105] Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. pages Volume 7, Issue 1, 1995.

[106] Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Computation*, 8(3):643–674, 1996.

[107] Ken C. Pohlmann. *Principles of Digital Audio.* McGraw-Hill, USA, 2005.

[108] Michiel Hazewinkel. "linear interpolation", encyclopedia of mathematics. 2001.

[109] Kanishka Tyagi, Son Nguyen, Rohit Rawat, and Michael Manry. Second order training and sizing for the multilayer perceptron. *Neural Processing Letters*, 51, 10 2019.

[110] Y. Ho and R. L. Kashyap. An algorithm for linear inequalities and its applications. *IEEE Transactions on Electronic Computers*, EC-14(5):683–688, 1965.

[111] Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, December 1998.

[112] Kang;Goldbaum Michael Kermany, Daniel;Zhang. Identifying medical diagnoses and treatable diseases by image-based deep learning. *The cell press journal*, pages 1122–1131, 02 2018.

[113] Yuval Netzer, Tiejie Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[114] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Chapter 11

Appendices

11.1 Network structure and training used for results

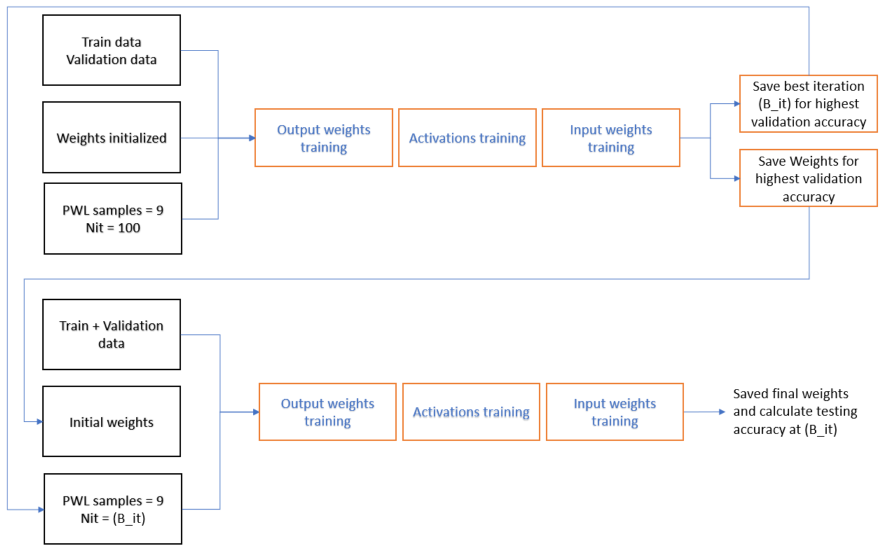


Figure 11.1: Shallow CNN with Linear classifier

11.2 Output reset

Classifiers with linear output activations usually suffer with a problem of inconsistent errors $(y_p(i) - t_p(i))$ that move in the direction opposite to that the probability of classification error P_e. This usually occurs when $y_p(i_c) \geq t_p(i_c))$ for the correct class i_c or $y_p(i_d) \leq t_p(i_d))$ for the inccorrect correct class i_d [109]. This problem is solved by developing new target outputs $t_p^{'}(i)$ not the actual class while keeping the constraint where the target margin satisfies $t_p^{'}(i_c) - t_p^{'}(i_d) \geq 1$. the new error function ([54],[55]) is defined as follows

$$E^{'} = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p^{'}(i) - y_p(i)]^2 \tag{11.1}$$

where new target output $t_p^{'}(i)$ is defined as

$$t_p^{'}(i) = t_p(i) + a_p + d_p(i) \tag{11.2}$$

and where a_p and $d_p(i)$ are initially set to zero. Since a_p is the same for each class, it has no effect on the percentage of error. In [54],[55], calculating the closed form expression for a_p. We find, derivative of new error equation with respect to a_p as,

$$a_p = \frac{1}{M} \sum_{i=1}^{M} [y_p(i) - t_p^{'}(i) - d_p(i)] \tag{11.3}$$

Similarly, $d_p(i)$ is defined as

$$d_p(i) = y_p(i) - t_p^{'}(i) - a_p \tag{11.4}$$

where, $d_p(i_c) \geq 0$ and $d_p(i_d) \leq 0$. a_p and $d_p(i)$ are not included during testing as it uses correct class $i_c(p)$. Therefore we include these parameters in $t_p^{'}(i)$ during training.
To avoid inconsistent errors we need $t_p^{'}(i_c) - y_p(i_c)$ and $t_p^{'}(i_d) - y_p(i_d)$ so

$$d_p(i_c) = y_p(i_c) - a_p - t_p(i_c)]u(y_p(i_c) - a_p - t_p(i_c)) \tag{11.5}$$

and

$$d_p(i_d) = y_p(i_d) - a_p - t_p(i_d)]u(t_p(i_d) - a_p - y_p(i_d)) \tag{11.6}$$

where $u(\cdot)$ denotes unit step function. Similar to inconsistent error problems, there are consistent errors $y_p(i) - t_p(i)$ move in the same direction as P_e. This usually occurs when $y_p(i_c) \leq t_p(i_c))$ or $y_p(i_d) \geq t_p(i_d))$ for any incorrect class i_d. The above problem is reduced because

$$\lim_{y_p(i_c) \to \infty} (y_p^{'}(i_c) - t_p(i_c)) = 0 \tag{11.7}$$

and similarly

$$\lim_{y_p(i_d) \to \infty} (y_p^{'}(i_d) - t_p(i_d)) = 0 \tag{11.8}$$

The OR algorithm is a multi-class version of Ho–Kashyap[110]

11.3 Data information

MNIST data

The digits datra used in this paper is taken from the MNIST data set[111], which itseld was constructed by modifying a subset of the much larger dataset produced by NIST(the National Institute of Standards and Technology). It comprises a training set of 60,000 examples and a test set of 10,000 examples. The original NIST data had binary (black or white) pixels. To create MNIST,these images were size normalized to fit in a 20×20 pixel box while preserving their aspect ratio. As a consequence of the anti-aliasing used to change the resolution of the images, the resulting MNIST digits are grey scale. These images were then centered in a 28 × 28 box. This dataset is a classic within the machine learning community and has been extensively studied.

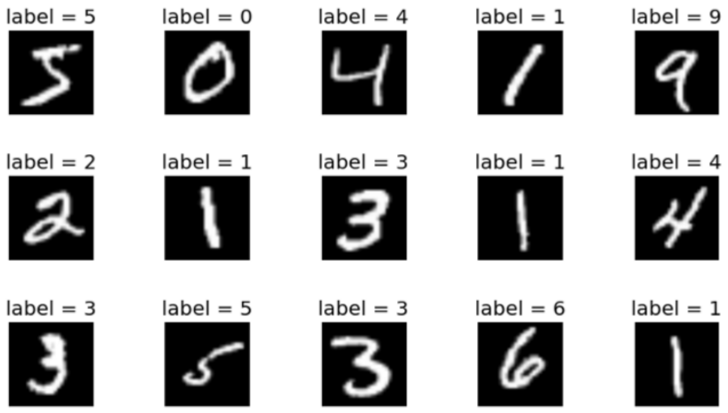


Figure 11.2: MNIST data example

Cifar10 data

The CIFAR-10 dataset [50] consists of 60,000 32 × 32 colour images in 10 classes, with 6000 images per class. There are 50,000 training images and 10,000 test images. The dataset is divided into five training batches and one test batch, each with 10,000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

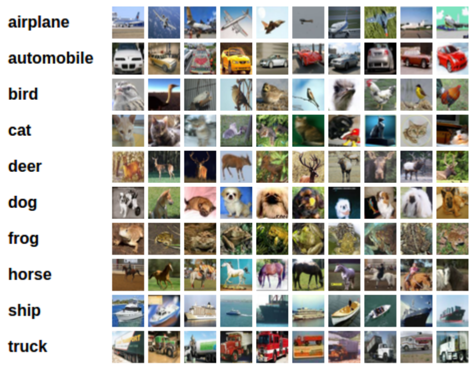


Figure 11.3: Cifar10 data example

Cifar100 data

The CIFAR-100 dataset [50] consists of 60,000 32 × 32 colour images in 100 classes with 600 images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. Each image comes with a ”fine” label (the class to which it belongs) and a ”coarse” label (the superclass to which it belongs).There are 500 training images and 100 testing images per class.

Superclass	**Classes**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Figure 11.4: Cifar100 data example

Cast Wrought data

The original Cast Wrought datasets consist of 16179 128 x 128 black and white images in 2 classes. The images are scaled down to 28 x 28 black and white images. The data is being given to the image processing and Neural Network lab by scrap classification company in Fortworth, Texas.

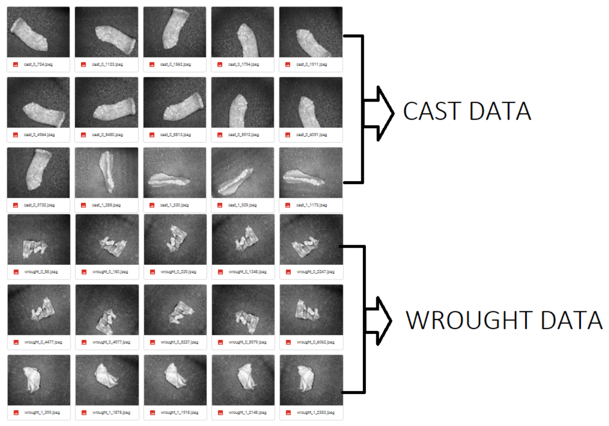


Figure 11.5: Cast wrought data example

Coin vs scrap data

The original coin scrap datasets consist of 1321 750 x 750 color images in 2 classes. The images are scaled down to 28 x 28. The data is being given to the image processing and Neural Network lab by scrap classification company in Fortworth, Texas.

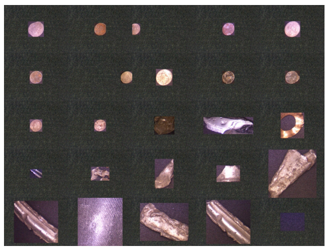


Figure 11.6: Cast wrought data example

Retinal OCT data

Retinal optical coherence tomography (OCT)[112] is an imaging technique used to capture high-resolution cross sections of the retinas of living patients. Approximately 30 million OCT scans are performed each year, and the analysis and interpretation of these images takes up a significant amount of time (Swanson and Fujimoto, 2017). The reference can be found in https://www.cell.com /cell/fulltext/S0092-8674(18)30154-5 There are 84,495 X-Ray images (JPEG) with 512 x 496 size and 4 categories (NORMAL,CNV,DME,DRUSEN).

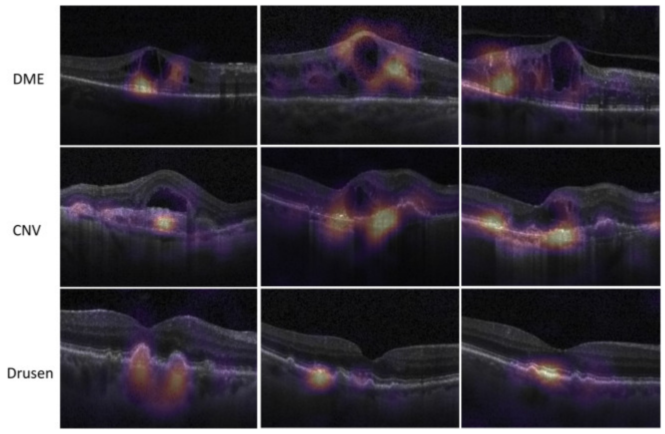


Figure 11.7: Cast wrought data example

Street View House Numbers (SVHN) Dataset

The Google street view housing numbers (SVHN) [113] is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images. Character level ground truth in an MNIST-like format. All digits have been resized to a fixed resolution of 32-by-32 pixels. The original character bounding boxes are extended in the appropriate dimension to become square windows, so that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions. Nevertheless this prepossessing introduces some distracting digits to the sides of the digit of interest. Loading the .mat files creates 2 variables: X which is a 4-D matrix containing the images, and y which is a vector of class labels. To access the images, X(:,:,:,i) gives the i-th 32-by-32 RGB image, with class label y(i).

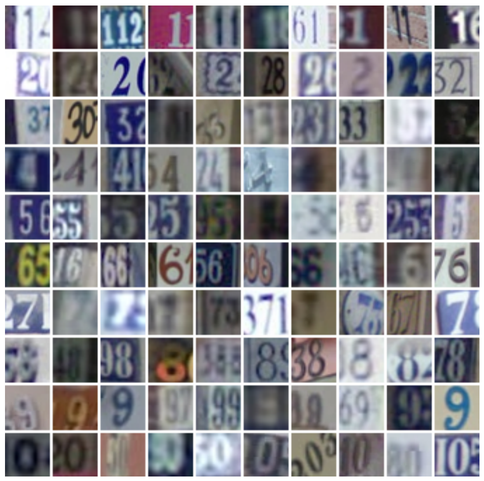
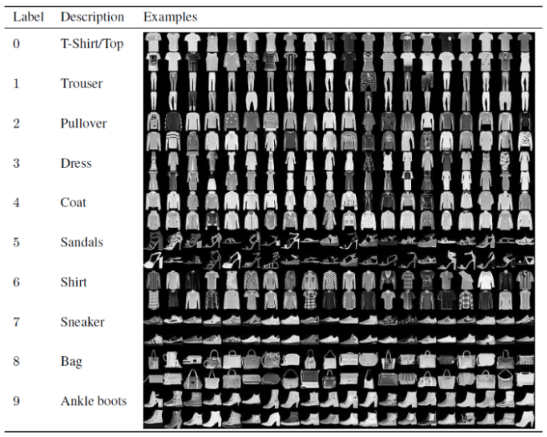


Figure 11.8: Svhn cropped data example

Fashion-MNIST data

The fashion-MNIST data[114] consists of 60,000 images in the training data with 10 categories and a test-set of 10,000 images. Each example is a 28x28 grayscale image. The authors intended Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.



Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 11.9: Fashion mnist data example

Intel Image classification data

Intel Image classification data consists of around 25k images of size 150x150 distributed under 6 categories. 'buildings' - 0,'forest' - 1,'glacier' - 2, 'mountain' - 3, 'sea' - 4,'street' - 5 . There are around 14k images in Train, 3k in Test and 7k in Prediction.This data was initially published on https://datahack.analyticsvidhya.com by Intel to host a Image classification Challenge.

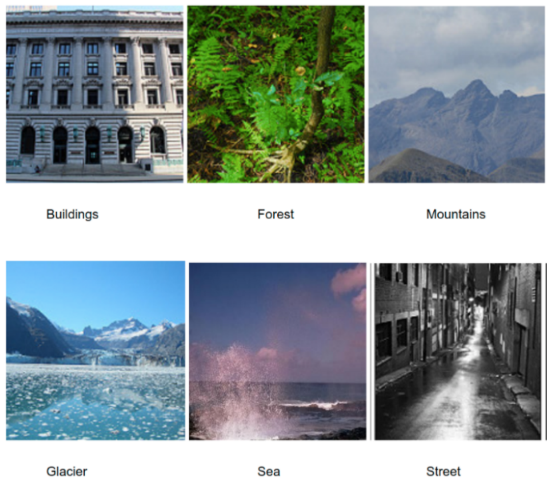


Figure 11.10: Intel image data example