# Some Studies for Rotational Invariance in Convolutional Neural Networks

**DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF**

Master of Technology
in
Computer Science

by


**Yash M. Sawant**

[ Roll No: CS-1614 ]


under the guidance of


**Dr. Swagatam Das**

Associate Professor
Electronics and Communication Sciences Unit




**Indian Statistical Institute**
**Kolkata-700108, India**

**July 2018**

# CERTIFICATE

This is to certify that the dissertation entitled **"Some studies for Rotational Invariance in Convolutional Neural Network "** submitted by **Yash M. Sawant** to Indian Statistical Institute, Kolkata, in partial fulfillment for the award of the degree of **Master of Technology in Computer Science** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

---

**Prof. Swagatam Das**
Associate Professor,
Electronics and Communication Sciences Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

# Acknowledgements

I would like to express my thankfulness to my supervisor Prof. Swagatam Das for encouraging me to pursue research in deep learning and for his helpful suggestion and constant guidance and support.

My sincere thanks to Shounak Datta for his valuable suggestions and discussions. I would also like to thank my friends, who are keen to listen to my doubts.

Lastly but not the least, I am thankful to God in the form of beautiful family, good teachers, and amazing friends.

# Abstract

Computer vision is an amazing field of using computing machinery to resemble human vision system where common tasks include image recognition. Convolutional Neural Networks(CNNs) has shown amazing capabilities for such tasks. Besides several advantages, CNNs are significantly susceptible to the rotational transformation of images. CNNs from the input images tries to abstract each image by learning local knowledge using convolutional filters applied over all parts of the image. We propose two methods for considering rotational invariance in learning. The first method uses rotational invariant filter method. The second method uses nonlinear neighbourhood component analysis for learning specialized filters for which the two images of the same class but in different rotational space lie close to each other. We discuss future scope solution for the NCA method.

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Convolutional Neural Network

Convolutional Neural Network architectures popularly known as CNN or ConvNet is a class of deep feed-forward neural networks mostly used for image recognition tasks.

CNNs like traditional feed-forward neural architectures consists of an input image, a convolutional hidden layer and output layer. The hidden layer of a CNN typically consists of convolutional layers, pooling layers, fully connected layers and sometimes normalization layers. One of the important operation in CNN is convolution and the so the name. Mathematically it is a cross-correlation rather than a convolution. A set of convolution filter is applied over the image computing a singular value for each part known as receptive field of the singular value. Doing so, the filters learn the spatial constancy feature detecting structures as a real-world image may be composed of a different repeating pattern. The filters are not learned in rotational invariant fashion as the filters are applied over the image as a sliding window stride horizontally and vertically.

CNNs also consists of two major pooling techniques for convolutional hidden volumes. They are used to generalize the local detecting feature in the vicinity and also helps in reducing the number of neurons for further processing. Pooling helps in resolving small translational invariance. Then comes the Fully Connected Layers(FCs) that connect every neuron in the last convolutional volume to every neuron in this layer similar to traditional MLP neural architecture. As the filters are mainly learned in the described fashion

reducing the classification loss, the CNNs are invariant to small distortions, translations, scaling but are sensitive to rotations. In this work, CNN is mostly adapted for achieving rotational invariance learning.

## 1.2 Rotational Invariance

In computer vision or image recognition tasks, CNN has outperformed traditional techniques such as SIFT[1] as compared in ImageNet classification [2]. Rotational Invariance is necessary for domains such as in textured images or in areas of application of object detection. Besides several feature learning advantages, CNNs are significantly susceptible to the rotational transformation. Rotational Invariance is necessary in many areas of computer vision such as object tracking, texture classification.

## 1.3 Our Contribution

In this work, a baseline CNN is compared to the proposed method for achieving rotational invariance. The proposed method is motivated by the work of Cheng et. al [3]. In the work of Cheng, rotational invariance is achieved through data augmentation of rotational transformed images with the original ones and adding a new FC layer which they called rotational invariant (RI) layer and with modified objective forced on this layer to achieve rotational invariance.

## 1.4 Outline

The rest of this report is organized as follows.

- Chapter 2 discusses related work for achieving rotational invariance for CNN architectures.
- Chapter 3 discusses some background for CNN and related concepts in this work.
- Chapter 4 discusses baseline CNN model and the proposed methods.
- Chapter 5 discusses results achieved by the baseline CNN, the proposed method and the future scope.

# Chapter 2

# Related Work

## 2.1 Data Augmentation with Rotational Invariant Layer

Due to limited size of the training set, performing data augmentation to artificially increase the number of training examples is necessary to avoid overfitting. Rotate[1] the image set randomly by defining K rotation angles.

$$T = \{\phi_1, \phi_2, \phi_3, ..., \phi_K\} \tag{2.1}$$

Given the training samples $X^{'} = \{x_i \in X \cup T_{\phi_i} X\}$ and their corresponding labels $Y^{'} = \{y_{x_i} | x_i \in X^{'}\}$.

In the original paper[3], CNN is modified by adding one additional layer known as Rotational Invariant layer with the modified objective

$$J = C(X^{'}, Y^{'}) + \lambda_1 R(X, T_\phi X) + \frac{\lambda_2}{2} \|W\|_2^2 \tag{2.2}$$

and

$$R(X, T_\phi X) = \frac{1}{2N} \sum_{x_i \in X} \|f(x_i|W) - \frac{1}{K} \sum_{j=1}^{K} f(T_{\phi_j} x_i|W)\|_2^2 \tag{2.3}$$

[1]Content taken from original paper (Cheng et.al 2016)

## 2.2 Rotation of Filters and Feature Maps with Maxout Networks

In [4], filter rotation is done through various angles such as can be of

$$T = \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}, \frac{7\pi}{4}\} \tag{2.4}$$

Thereby producing convolution volumes equal to the number of rotated angles. These convolutional volumes can individually be treated as single ConvNet flow[5] for each different rotated angles and can be concatenated at the end to create a single representation layer for all the channel by using Maxout[6] technique suggested by (Goodfellow, 2013). In practice, the number of output feature maps will be large and for rotated angles will increase furthermore the number of feature maps and following convolutional layer will receive the higher number of feature maps. In order to reduce these network requirement, Maxout is carried on different rotated feature maps itself to concatenated into one feature map and sent further for next convolutional layer. The key idea of maxout networks is to output the max of several input feature maps when applied to several feature maps of the same dimension, will output a single feature map of the same original dimension.

In [4] filters are rotated in the following way.

| | | | |
|---|---|---|---|
| 1 2 3<br>4 5 6<br>7 8 9 | 4 1 2<br>7 5 3<br>8 9 6 | 7 4 1<br>8 5 2<br>9 6 3 | 8 7 4<br>9 5 1<br>6 3 2 |
| 9 8 7<br>6 5 4<br>3 2 1 | 6 9 8<br>3 5 7<br>2 1 4 | 3 6 9<br>2 5 8<br>1 4 7 | 2 3 6<br>1 5 9<br>4 7 8 |

Figure 2.1: Rotating $3 \times 3$ Filters by angles in set T

The corresponding weight values are shifted to right in circular fashion as shown in figure 2.1 for 0, $\frac{\pi}{4}$ and so on.

# Chapter 3

# Preliminaries

## 3.1 Restricted Boltzmann Machine (RBM)

The restricted boltzmann machine (RBM) is a two-layer, bipartite, undirected graphical model with a set of binary hidden units $h$, a set of (binary or real-valued) visible units $v$, and a symmetric connections between these two layers represented by a weight matrix $W$. The probabilistic configuration for an RBM is defined by its energy function as follows:

$$P(v,h) = \frac{1}{Z} exp(-E(v,h)) \tag{3.1}$$

where Z is the partition function. If the visible units are binary, $E(v,h)$ is defined as:

$$E(v,h) = -\sum_{i,j} v_i W_{ij} h_j - \sum_j b_j h_j - \sum_i c_i v_i \tag{3.2}$$

### 3.1.1 Training RBM

The parameters of the RBM can be learnt by performing stochastic gradient descent on the empirical negative log-likelihood[1] of the training data.

$$\mathcal{L}(\theta, \mathcal{D}) = \frac{1}{N} \sum_{x^{(i)} \in \mathcal{D}} \log p(x^{(i)}) \tag{3.3}$$

[1]Content taken from `deeplearning.net/tutorial/rbm.html`

$$P(x) = \sum_h P(x,h) = \sum_h \frac{exp(-E(x,h))}{Z} \tag{3.4}$$

And, free energy for convenience is defined as follows:

$$\mathcal{F}(x) = -\log \sum_h exp(-E(x,h)) \tag{3.5}$$

The derivative of negative of $\log p(x)$ is given by:

$$-\frac{\partial \log p(x)}{\partial \theta} = \frac{\partial \mathcal{F}(x)}{\partial \theta} - \sum_{\bar{x}} p(\bar{x}) \frac{\partial \mathcal{F}(\bar{x})}{\partial \theta} \tag{3.6}$$

We can approximate the second term in equation 3.6 by Monte Carlo Markov Chain (MCMC) technique

$$\sum_{\bar{x}} p(\bar{x}) \frac{\partial \mathcal{F}(\bar{x})}{\partial \theta} = \frac{1}{|\mathcal{N}|} \sum_{\bar{x} \in \mathcal{N}} \frac{\partial \mathcal{F}(\bar{x})}{\partial \theta} \tag{3.7}$$

where $\bar{x}$ of $\mathcal{N}$ to be sampled from $p(x)$.

### 3.1.2 Sampling in an RBM

Gibbs sampling is the technique used for generating samples from $p(x)$ by initializing the visible variables $v$ by some $x^{(i)} \in \mathcal{D}$.

- Gibbs Sampling The idea in Gibbs sampling is to generate posterior samples by sweeping through each variable (or block of variables) to sample from its conditional distribution with the remaining variables fixed to their current values.

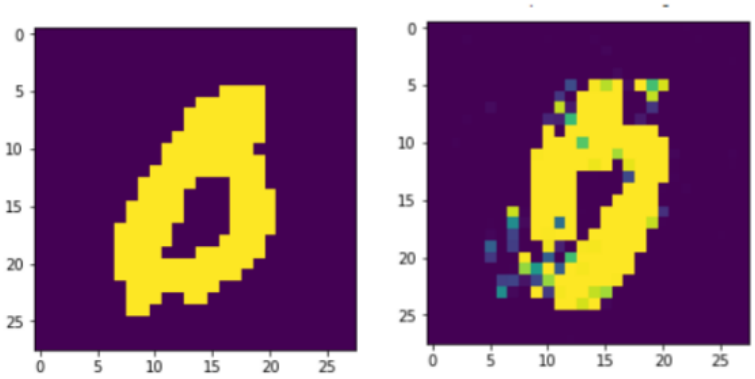
$$h^{(n+1)} \sim sigm(W^{'} v^{(n)} + c) \tag{3.8}$$

$$v^{(n+1)} \sim sigm(W h^{(n+1)} + b) \tag{3.9}$$

  As individual units within a layer are independent therefore we have two main conditional equations where we can sample the variables easily. As in Gibbs sampling, we repeatedly generate sample value from conditional equations (3.8 & 3.9) till convergence.

- **Contrastive Divergence (CD - K)** : We start the Markov chain for each $x^{(i)} \in \mathcal{D}$ and repeatedly apply Gibbs sampling for K steps and apply the update using (3.7).
- **Persistent CD** : We divide the data $\mathcal{D}$ in mini-batch $\mathcal{M}$ and for each mini-batch we start the markov chain with some $x^{(i)} \in \mathcal{M}^{(j)}$ for jth mini-batch and for each update we use previously computed sample for generating new samples.

For MNIST image of 784 visible units and 500 hidden units. Figure 3.1 shows one of input image



(a) Input Image (b) Reconstruction

Figure 3.1: RBM reconstruction for (784, 500) model

## 3.2 Non-Linear Neighbourhood Component Analysis

The idea of Non-Linear Neighbourhood Component Analysis (Nonlinear NCA) is transforming non linearly input space to a low dimensional feature space in which K-Nearest neighbour classification performs well. The paper[7] described a very effective and unsupervised way of training a multi-layer, non-linear "encoder" network that transforms the input data vector $\mathbf{x}$ into a low-dimensional feature representation $f(\mathbf{x}|W)$ that captures a lot of the

structure in the input data.

There are two stages of training the network.

- Pretraining : Initialize the parameter vector $W$ that defines the mapping from input vector to their low-dimensional representation.
- Fine-tuning : The parameter $W$ can be fine-tuned by performing gradient descent in the NCA objective function

### 3.2.1 Nonlinear NCA

We[2] are given a set of N labelled training cases $(\mathbf{x}^a, c^a)$. For each training vector $\mathbf{x}^a$, define the probability that point $a$ selects one of its neighbours $b$ in the transformed feature space as:

$$p_{ab} = \frac{exp(-d_{ab})}{\sum_{z \neq a} exp(-d_{az})} \tag{3.10}$$

$$d_{ab} = \|f(x^a|W) - f(x^b|W)\|^2 \tag{3.11}$$

The probability that point a belongs to class k depends on the relative proximity of all other data points that belong to class $k$:

$$p(c^a = k) = \sum_{b:c^b=k} p_{ab} \tag{3.12}$$

The NCA objective is to maximize the expected number of correctly classified points on the training data:

$$O_{NCA} = \sum_{a=1}^{N} \sum_{b:c^a=c^b} p_{ab} \tag{3.13}$$

[2]Content taken from original paper [7]

Figure 3.2[7] shows how two different images can be non-linearly transformed to feature space where input images of same class can be make to lie close to each other.

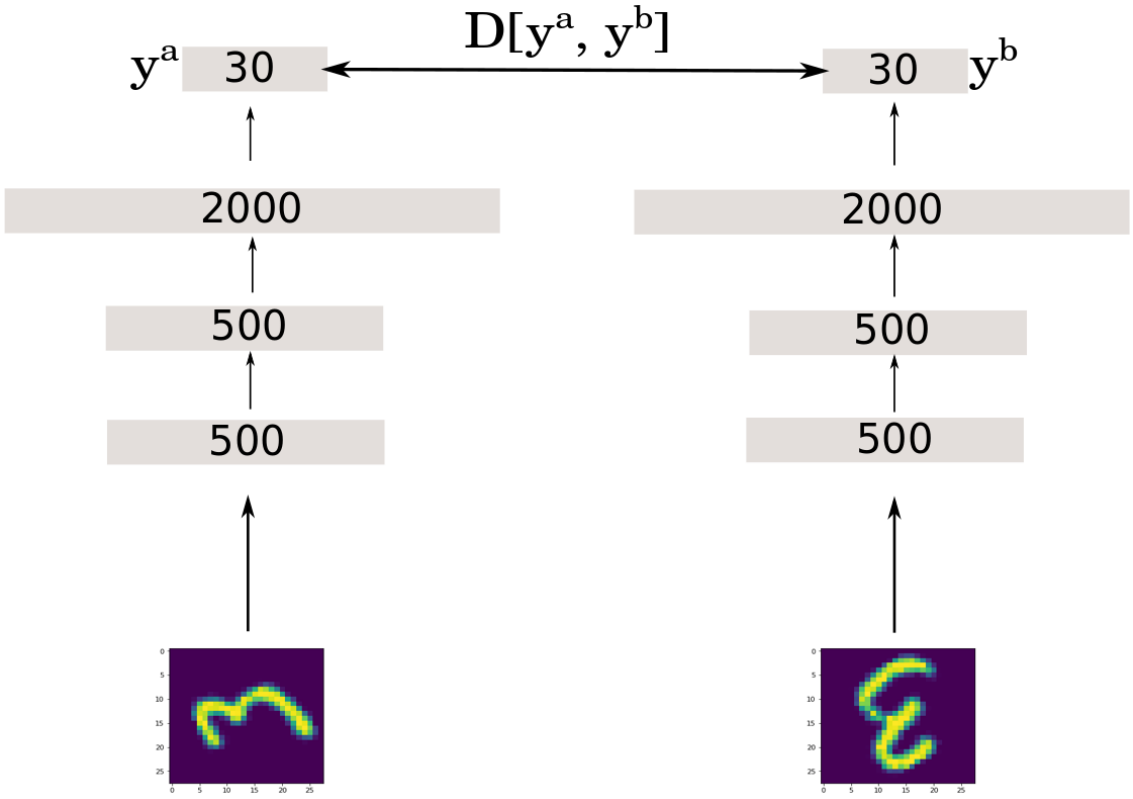



Figure 3.2: Learning a representation in low dimensional feature space by non-linear transformation from images where similar class input images lie close to each other.

### 3.2.2 Deep Belief Network

In the figure 3.2, input image is transformed to low dimensional feature space. In the network, we can define joint probability distribution of the network as follow

$$p(\mathbf{x}, \mathbf{h}^{(1)}, \mathbf{h}^{(2)}, \mathbf{h}^{(3)}, \mathbf{y}) = p(\mathbf{h}^{(3)}, \mathbf{y})\ p(\mathbf{h}^{2}|\mathbf{h}^{(3)})\ p(\mathbf{h}^{(1)}|\mathbf{h}^{(2)})\ p(\mathbf{x}|\mathbf{h}^{(1)}) \quad (3.14)$$

### 3.2.3 Greedy Layer-wise Pretraining

To [8] obtain a estimator of the gradient of the log-likelihood of an RBM(Equation 3.7), the loglikelihood of a value $\mathbf{x}$ under the model of the RBM is

$$\log p(\mathbf{x}) = \log \sum_{\mathbf{h}} p(\mathbf{x}, \mathbf{h}) = \log \sum_{\mathbf{h}} exp(-E(\mathbf{x}, \mathbf{h})) - log \sum_{\tilde{\mathbf{x}}, \mathbf{h}} exp(-E(\tilde{\mathbf{x}}, \mathbf{h}) \tag{3.15}$$

Concavity of the log function satisfies

$$\log(\sum_i \lambda_i a_i) \geq \sum_i \lambda_i \log(a_i) \tag{3.16}$$

where $\sum_i \lambda_i = 1 \& \lambda_i \geq 0$

For any model $p(\mathbf{x}, \mathbf{h}^{(1)})$ with latent variables $\mathbf{h}^{(1)}$ we can write:

$$\log p(\mathbf{x}) = \log \left( \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \tag{3.17}$$

By equation (3.16), we can write equation (3.17) as

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log \left( \frac{p(\mathbf{x}, \mathbf{h}^{(1)})}{q(\mathbf{h}^{(1)}|\mathbf{x})} \right) \tag{3.18}$$

Considering $q(\mathbf{h}^{(1)}|\mathbf{x})$ as $\lambda_i$. This is called variational bound.

$$\log p(\mathbf{x}) \geq \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{x}, \mathbf{h}^{(1)}) - \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log(q(\mathbf{h}^{(1)}|\mathbf{x}) \tag{3.19}$$

- The first term in equation 3.19 if maximized will maximize approximately $\log p(\mathbf{x})$
- $\log p(\mathbf{x}, \mathbf{h}^{(1)})$ can be written as

$$\log p(\mathbf{x}, \mathbf{h}^{(1)}) = \log p(\mathbf{x}|\mathbf{h}^{(1)}) + \log p(\mathbf{h}^{(1)}) \tag{3.20}$$

- If we try to minimize the negative of

$$\sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log p(\mathbf{h}^{(1)}) = \sum_{\mathbf{h}^{(1)}} q(\mathbf{h}^{(1)}|\mathbf{x}) \log \sum_{\mathbf{h}^{(2)}} p(\mathbf{h}^{(1)}, \mathbf{h}^{(2)}) \qquad (3.21)$$

  It will result in maximizing the lower bound of $\log p(\mathbf{x})$ by (3.18) and hence $\log p(\mathbf{x})$
  Equation (3.21) can be approximated by using first learning the first layer RBM and sample $\mathbf{h}^{(1)}$ from $\mathbf{x}$.

The following procedure is used to pretrain the DBN.

- Learn the parameter $W^1$ of a model by considering it as RBM.
- Freeze the parameters of the lower-level model and treat the hidden features activation by sampling each binary features from visible units, as the data for the next layer model.
- Proceed recursively for all the layers.

# Chapter 4

# Proposed Methods

## 4.1 Baseline CNN

A Baseline CNN consists of two convolutional layer using $5 \times 5$ filters. For each layer ReLU activation is used. It also consists of 5 FCs layers.

| Layer | # of Units | Kernel/Pooling Size | Stride | # of Parameters |
|---|---|---|---|---|
| Conv2D_Conv1 | 32 | (5, 5) | (1, 1) | 800 |
| MaxPoo2D_Conv1 | - | (2, 2) | (2, 2) | 0 |
| Conv2D_Conv2 | 64 | (5, 5) | (1, 1) | 51200 |
| MaxPool2D_Conv2 | - | (2, 2) | (2, 2) | 0 |
| Conv2_Flat | - | - | - | 0 |
| FullyConnected(FC1) | 3136 | - | - | 3201024 |
| FullyConnected(FC2) | 1024 | - | - | 716800 |
| FullyConnected(NCA1) | 700 | - | - | 210000 |
| FullyConnected(NCA2) | 300 | - | - | 21000 |
| FullyConnected(NCA3) | 70 | - | - | 630 |

Table 4.1: Baseline CNN

Table 4.1 describes each layer for the architecture shown in Figure 4.1, the first column states the layer type, the second column shows the corresponding number of units. For example, Conv2D_Conv1 contains 32 feature maps of same size as that of input image. The Kernel/Filter size used is (5, 5) and stride of 1 horizontally and 1 vertically as described in table (1, 1). As the

number of filters is 32 of size $5 \times 5$, we have total number of parameters of the 1st layer $5 \times 5 \times 32$. For every layer ReLU activation functions are used. NCA layers are same as FullyConnected Layer but as described in Chapter 3, the features computed from the above layer are projected onto these layer to make same class input images lie close to each other in the NCA3 layer projected space.

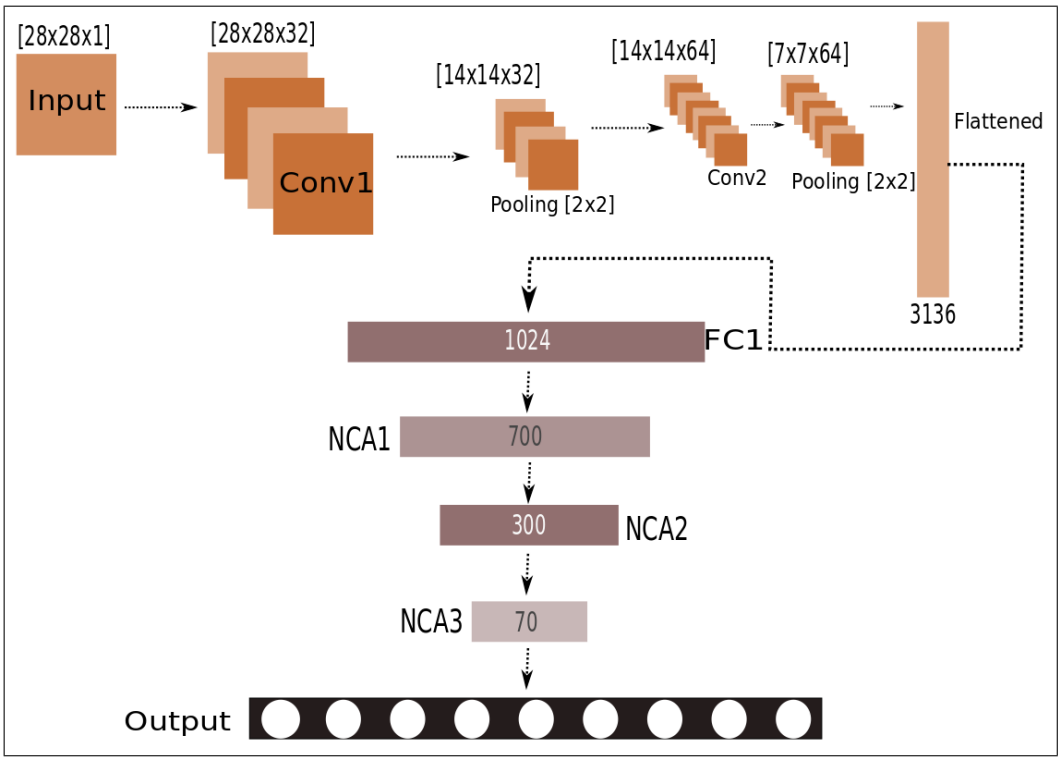
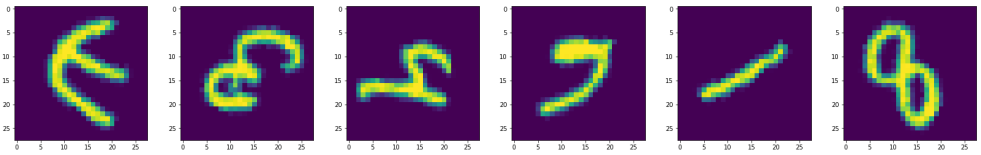



Figure 4.1: Baseline CNN for Comparison

### 4.1.1 Training

Figure 4.1 shows the architecture of CNN for MNIST[9] dataset, each input image is rotated randomly by angle multiple of $\frac{\pi}{4}$. Figure 4.2 shows some of the rotated MNIST images.

Image (d) is labelled 6 and also images labelled 6 & 9 are same in rotated space and hence treated in class labelled 6. Therefore, output layer consists of 9 softmax units.



(a) Label 7 (b) Label 3 (c) Label 4 (d) Label 6 (e) Label 1 (f) Label 8

Figure 4.2: Rotated MNIST images

## 4.2 Rotational Invariant Filters

CNN has two main components of learning features in the network. Convolution and pooling out essential features within a local region of interest. If we suppose these filters to be rotational invariant that is for same input image, each of the rotated filters result in approximately same feature activation in a layer of interest. In table 4.1 layer FC2 is forced to be close to each other for each of the rotated filter by using equation 4.1 where FC2 output is denoted by $f_c(x^{(i)}|W)$ where $c$ is described rotation angle and output results from rotating filter by angle described by $c$. For MNIST training, the rotating angles are

$$T = \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}, \frac{7\pi}{4}\} \tag{4.1}$$

$$O_{RIF} = \sum_{i=1}^{N} \sum_{1<a<b\leq K} \|f_a(x^{(i)}|W) - f_b(x^{(i)}|W)\|^2 \tag{4.2}$$

Here, K is the number of defined rotation angles for the filters.

### 4.2.1 Training Rotational Invariant Filters

Equation 4.2 forces non-linearly the filters to have nearly same activation in the layer of interest.

For MNIST dataset training is done as follows:

- In CNN described in Table 4.1, network is trained for MNIST images without any rotation with the objective

$$O = C(X, Y) + O_{RIF} \tag{4.3}$$

$C(X, Y)$ is cross-entropy loss and $O_{RIF}$ is equation 4.2 rotation invariant filter objective.

- After the parameter update $W$, mini-batch is rotated and parameters belonging to layers after FC2 are updated since rotation of image involves corresponding rotation in space of FC1 and only parameters after layer FC2 which is enforced in equation 4.2 needs to be fine-tuned.

## 4.3 NCA Objective on CNN

As described in Chapter 2, Neighbourhood component analysis objective function learns low dimensional feature space where KNN perform well. We can extend this concept to CNN by forcing NCA3 in table 4.1 to be close for each of similar class images.

### 4.3.1 Difficulties in Training

For all the layers in Table 4.1 ReLU activation is used. NCA3 is 70 sized vector where we are applying equation (3.13). Generally outputs from ReLU activation goes larger as we go deeper into the layer such as for NCA3.

$$p_{ab} = \frac{exp(-\|f(x^a|W) - f(x^b|W)\|^2)}{\sum_{z \neq a} exp(-\|f(x^a|W) - f(x^b|W)\|^2)} \tag{4.4}$$

If in the numerator term in Equation 4.4, the L2 norm of the difference of NCA3 vector for two points $a$ and $b$ is large, the $p_{ab}$ is very small and hence insignificant for computing gradients. We can L2 normalize the NCA vectors and use it for computing $p_{ab}$. Still we found out that L2 normalize is not enough for getting significant gradients for weight updation. In the future scope, the possible solution for this is discussed. For the cross-entropy, gradients for the first layer filter weights are of order 1e5 and for the NCA, the corresponding gradients are 1e-3.

# Chapter 5

# Results and Experiments

## 5.1 Preparing Datasets

We performed all our experiments using MNIST datasets. We performed rotation of MNIST images by angle multiple of $\frac{\pi}{4}$ as described in the chapter 4 - section 4.1.1.

## 5.2 Results

For the rotational invariant filter, we can do the following two procedures.

**Procedure I** is described as follows:

1. Feed input images(MNIST) in mini-batches $\mathcal{M}$ without any rotation and apply updates based on cross-entropy loss and $O_{RIF}$ as described in Equation (4.2).

2. For some number of mini-batches, feed MNIST images with rotation and apply updates for the parameters belonging to layers after FC2 in table 4.1

**Procedure II** is described as follows:

1. For every mini-batch $\mathcal{M}^{(i)}$. Apply the updates as in step 1 of procedure I.

2. For the mini-batch $\mathcal{M}^{(i)}$ apply the rotation and feed the rotated mini-batch for the updation of parameters belonging to layers after FC2 in table 4.1

CLOSS[1][1] is cross-entropy loss for softmax output used for augmentated(rotated) dataset(MNIST). CLOSS[2][2] is used for normal MNIST images datasets. RIF is rotational invariant filter enforced on FC2 and bracketed are the applied procedure discussed above. Each epochs is of 54 iterations and each mini-batch is of size 1000. Accuracy is calculated for 1000 rotated MNIST test images. In RIF(I) 10 mini-batches are applied. For RIF(II) accuracy is taken after 27 iterations as 1 iteration is equivalent to 2 iterations in case of CLOSS. Table 5.1 shows the result achieved for MNIST datasets

| Method | # of Epoch(s) | Accuracy |
|---|---|---|
| CLOSS[1] | 1 | 528/1000 |
| | 2 | 622/1000 |
| | 3 | 675/1000 |
| | 4 | 701/1000 |
| | 5 | 732/1000 |
| | 6 | 750/1000 |

Table 5.1: Results achieved by Baseline CNN for MNIST datasets

[1]Cross Entropy Loss on Augmentated datasets

[2]Cross Entropy loss without rotation for weight updation of convolution filters

| # of Epoch(s) | CLOSS[2] | CLOSS[2] + RIF(I) | CLOSS[2] + RIF(II) |
|---|---|---|---|
| 1 | 368/1000 | **400**/1000 | 396/1000 |
| 2 | 448/1000 | **521**/1000 | 479/1000 |
| 3 | 507/1000 | 543/1000 | **544**/1000 |
| 4 | 512/1000 | **584**/1000 | 572/1000 |
| 5 | 547/1000 | 605/1000 | **617**/1000 |
| 6 | 570/1000 | 615/1000 | **639**/1000 |

Table 5.2: Results achieved by the Proposed Methods for MNIST datasets

# Chapter 6

# Conclusion and Future Work

## 6.1 Future Work

### 6.1.1 Pretraining CNN as Convolutional DBN

Convolutional Neural Network can be pretrained as convolutional deep belief network as (Lee[10], 2009) introduced the method for training the network.

### 6.1.2 Convolutional RBM

Convolutional RBM (CRBM)[1] is similar to RBM discussed in chapter 3, but the weights are shared among all location in the image as in CNNs. The basic CRBM consists of two layer: visible layer and hidden layer. As in CNN, the size of filter $N_W$ and size of image $N_V$, the size of output convolution hidden layer ($N_H = N_V - N_W + 1$)

### 6.1.3 Probabilistic Max-pooling

Suppose we have a model with a visible layer V and a hidden layer H and a pooling layer P. Hidden layer H is divided into $C \times C$ for each of filter $N_W^k$. Each block $B_\alpha$ can be sampled independently.

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{k=1}^{K} \sum_{i,j=1}^{N_H} \sum_{r,s=1}^{N_W} h_{ij}^k W_{rs}^k v_{i+r-1,j+s-1} - \sum_{k=1}^{K} b_k \sum_{i,j=1}^{N_H} h_{ij}^k - c \sum_{i,j=1}^{N_V} v_{ij}. \quad (6.1)$$

[1] Content taken from orginal paper [10]

Suppose $h_{ij}^k$ is a hidden unit in $B_\alpha$, the conditional probability

$$p(h_{ij}^k = 1|\mathbf{v}) = \frac{exp(-E(h_{ij}^k, \mathbf{v}))}{1 + \sum_{(p,q)\in B_\alpha} exp(-E(h_{pq}^k, \mathbf{v}))} \tag{6.2}$$

and the conditional probability for the corresponding pooling unit $p_\alpha^k$

$$p(p_\alpha^k = 0|\mathbf{v}) = \frac{1}{1 + \sum_{(p,q)\in B_\alpha} exp(-E(h_{pq}^k, \mathbf{v}))} \tag{6.3}$$

Sampling H given V are done using (Equation 5.2 and 5.3) and for V are done using weight values used for connection between V and H. Using the above procedure, we can pretrain CNN as stack of CRBM as discussed for the case of DBN in Chapter 3 as Convolutional Deep Belief Network. In [7] after pretrain, we can apply $O_{NCA}$ objective for fine-tuning the network.

## 6.2 Conclusion

In this work, two methods were discussed Rotational Invariant Filter and Non-Linear NCA method for achieving rotational invariance in CNN. For the rotational invariant filter learning, an RI convolutional filter is discussed. RIF method can perform better if a careful rotation is applied to the filters. We had used simple bilinear image rotation transformation. Filters can also be rotated in the hard-coded manner as discussed in Chapter 2 - Figure 2.1. For the Non-Linear NCA, it is necessary for the CNN to be pre train first using CDBN for gradients of the NCA to be effective. Implementation is available at `https://github.com/yashmrsawant/RICNN`

# Bibliography


[1] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

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

[3] Gong Cheng, Peicheng Zhou, and Junwei Han. Learning rotation-invariant convolutional neural networks for object detection in vhr optical remote sensing images. *IEEE Transactions on Geoscience and Remote Sensing*, 54(12):7405–7415, 2016.

[4] Hongyang Gao and Shuiwang Ji. Efficient and invariant convolutional neural networks for dense prediction. *arXiv preprint arXiv:1711.09064*, 2017.

[5] Sander Dieleman, Kyle W Willett, and Joni Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly notices of the royal astronomical society*, 450(2):1441–1459, 2015.

[6] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[7] Ruslan Salakhutdinov and Geoff Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Artificial Intelligence and Statistics*, pages 412–419, 2007.

[8] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[9] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[10] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.