I.J. Image, Graphics and Signal Processing, **2016, 3, 19-27**
Published Online March 2016 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijigsp.2016.03.03

Pattern Recognition: Invariance Learning in Convolutional Auto Encoder Network

[1]Oyebade K. Oyedotun, [2]Kamil Dimililer
[1,2] Near East University, Lefkosa, via Mersin-10, North Cyprus
Email: [1]oyebade.oyedotun.k@ieee.org, [2]kamil.dimililer@neu.edu.tr

Abstract—The ability of the human visual processing system to accommodate and retain clear understanding or identification of patterns irrespective of their orientations is quite remarkable. Conversely, pattern invariance, a common problem in intelligent recognition systems is not one that can be overemphasized; obviously, one's definition of an intelligent system broadens considering the large variability with which the same patterns can occur. This research investigates and reviews the performance of convolutional networks, and its variant, convolutional auto encoder networks when tasked with recognition problems considering invariances such as translation, rotation, and scale. While, various patterns can be used to validate this query, handwritten Yoruba vowel characters have been used in this research. Databases of images containing patterns with constraints of interest are collected, processed, and used to train and simulate the designed networks. We provide extensive architectural and learning paradigms review of the considered networks, in view of how built-in invariance is learned. Lastly, we provide a comparative analysis of achieved error rates against back propagation neural networks, denoising auto encoder, stacked denoising auto encoder, and deep belief network.

Index Terms—Convolutional neural network, auto encoders, pattern invariance, character recognition, Yoruba vowel characters.

I. Introduction

Computer vision is an interdisciplinary field that deals with the analysis and understanding of acquired real world data images. It also involves machines that simulate the perception of images as in human ability and processing. The goal of computer vision is to make useful decisions about real physical objects and scenes based on sensed images [1]. Generally, the field of computer vision is not one that can be separated from image processing and machine learning.

Image processing usually involves operations or algorithms that condition images depending on the aims of applications. Some of the operations include image filtering, dimension reduction, enhancement, segmentation, and characteristics evaluation.

Machine learning, a field under artificial intelligence, deals with the design of adaptive systems that can learn and improve its performance over time due to acquired experiential knowledge. The fusion of image processing and machine learning can be considered the backbone of computer vision systems.

Furthermore, there has been a significant change in the task required of computer vision systems recently; a sway from what can be considered "heavy" image processing or feature extraction schemes and "simple" machine learning tasks, to "low" image processing or feature extraction schemes and "heavy" machine learning tasks. i.e. demanding more of machine learning in applications. Lately, it can be seen that machine learning systems and paradigms which can explore almost raw data have received significant research attention, of course, this is evident when we expand the definition of intelligence.

This research reviews some common and important constraints that occur in computer vision, pattern invariance, considering some neural networks which enjoy architectures inspired by the biological visual processing system. Intelligent recognition systems should be accommodating of moderate pattern invariances such as translation, rotation, and scale. Furthermore, it is desirable that such intelligent systems should have built-in structure and ability to cope and understand these invariances.

Convolutional neural network and convolutional auto encoder neural networks have been considered in this research for study, as to how their structures and learning algorithms affect the achievable level of built-in invariance.

To validate the query of this research, Yoruba vowel characters have been used as patterns. The remaining sections in this paper present the structure, learning algorithms, training, testing, and analysis of achieved error rates for the considered networks.

II. Literature Review

The three major approaches to the problem of invariant pattern recognition are discussed below.

(a) Invariant feature extraction approach: This approach involves the extraction of features which are insensitive to pattern invariance. The features extracted remain fairly consistent even when patterns are moderately distorted. The success of this approach lies in extracting features that are robust to moderate pattern invariance, as different extracted features yield different levels of invariant pattern recognition. i.e. some features

Copyright © 2016 MECS

are more stable or less sensitive to pattern invariance than others. Common techniques that are applied to invariant feature extraction include shape orientation, Fourier transforms, Wavelet transforms, Fourier-Mellin Descriptors, moment invariants, etc. [2] [3] [4] [5]. The extracted features are then used to train a classifier which learns the associative mapping of such features to the output classes.

(b) Machine learning approach: In this approach, invariant pattern recognition is achieved in the learning system. Generally, the intelligent system is task with learning features that associate patterns with their corresponding target classes, even after the distortion of such patterns. Neural networks have been shown to suffice for such situations. However, only relatively moderate pattern invariance is achievable in these conventional networks. To significantly improve the performance of these systems on invariant pattern recognition, the two major techniques used are briefly discussed below.

- Data manipulation: The intelligent system is trained with distorted copies of training examples of the invariant patterns; this allows the system to learn the association of such distorted patterns with the corresponding target classes [6]. This can be considered a trivial approach to achieving invariant pattern recognition [7]. It is obvious that the technique requires the collection of labelled distorted patterns for learning; this increases cost and the manual input required.
- Hard coding: In many works, invariant pattern recognition is achieved by configuring and constraining the structure and weights of neural networks in some fashion. This has proven quite successful in many works [8] [9].

(c) Hybrid approach: In other works, the first and second approaches discussed above are combined to achieve more invariant pattern recognition. Performances of neural networks with structures that are apt for pattern invariance learning are boosted by augmenting the original training data with manipulated data.

III. Convolutional Neural Network (CNN)

Convolutional neural networks benefit from some biologically inspired architectural build such as the concept of local receptive fields [10]; similarity in build can be associated with the neocognitron, by Fukushima [11]. These networks have built-in architectures that specially lend themselves to problems encountered in most computer vision systems. The characteristics of these networks leveraged on in computer vision are concepts of local connectivity, weight sharing, and sub-sampling or pooling. It is somewhat obvious that in real-life problems of computer vision, we often have to deal with high-dimensional images, thus requiring enormous network parameters for computation. Hence, the optimization of these parameters becomes a major constraint considering achievable error rates and associated costs of hardware suitable enough for performing such computations in reasonable time; also, required memory is yet another problem to be considered. Inasmuch as conventional feedforward networks can be drafted for this purpose, some structural constraints found in these networks make their adaptability for invariant pattern recognition and therefore use only second convolutional networks. Furthermore, it is observable that feedforward networks ignore the 2D topology of data as it is found in image applications. [12]. i.e. the training data elements or attributes can be offset consistently through a data set without affecting the performance of the network significantly. Conversely, images have attributes (pixel values) that are strongly local, as neighbouring pixels are usually related, hence, the need for architectures that better simulate the human visual processing of images.

Also, the architecture of convolutional neural networks makes possible the realization of some built-in invariance during the learning phase of these networks. The way attributes are learned from input images using local receptive fields, weight sharing and pooling operations incorporate a better understanding of features that make input images different from one another. A convolutional network can be "roughly" considered as a feedforward network with alternating convolution and pooling layers, while the last layer is usually a fully connected multi-layer network or any other classifier. The following subsections describe briefly convolution and pooling operations.

3.1 Convolution

The first convolution layer generally succeeds the input layer in a convolutional neural network, this layer can be viewed as 2D planes of units (neurons); each plane of units is called a feature map or plane.

Each feature map has units in 2D arrangement and these units share a common set of weights depending on the size of the receptive field.

The receptive field is a region (patch) of the input captured by each unit in any feature map; generally, a constant receptive field is used at any particular convolution layer. i.e. all feature maps for each unit have the same size of receptive field. The receptive field is measured in pixels along both axes of an input image. e.g. for an input image of size 64×64, the receptive field could be of size 9×9. The receptive field is usually considered as a filter or kernel which is used to convolve the whole input image for the convolution layer operation. It is also noteworthy that each feature map in the convolution layer extracts a distinct feature about the input image depending on the particular kernel that is applied on a feature map. Thus, it is possible that a feature map may extract horizontal edges, another extracts vertical edges, while still, another extracts points, etc. It can therefore be inferred that the number of different kernels used in the convolution layer will be the number of the feature maps in that particular convolution

Copyright © 2016 MECS

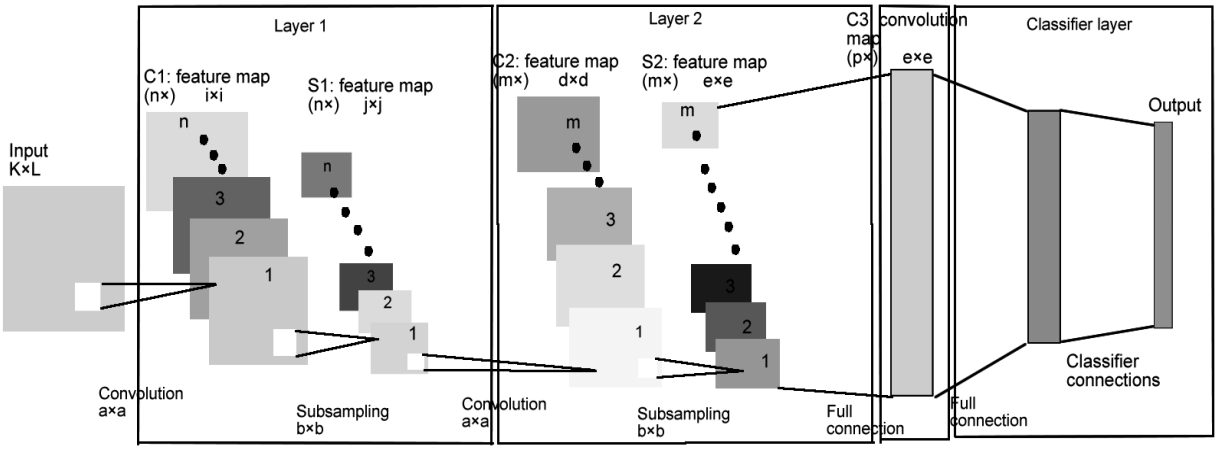


Fig.1. Convolutional neural network

layer. Also, all feature maps have the same number of units. All units in a feature map have interconnections to the input through the receptive field, with the number of interconnections being the size of the kernel used. e.g. if the kernel size is 4 × 4, then each unit has 16 interconnections. The values of the kernel used are the connection weights of units when the network has just been initialized.

The idea of convolving a distinct kernel all over the input image for a particular feature map is such that the same feature may be extracted all over the image for that feature map, and that parameter sharing can also be achieved. This operation also greatly reduces the number of trainable weights, and therefore computation that would have been required as is evident in conventional feedforward network; since, there is full interconnection between layers. In contrast, convolutional networks use local connectivity and parameter sharing.

To illustrate this significance, consider an input image of size 120 × 120, the typical feedforward network, with say 60 hidden neurons will require (14,400 × 60 + 60) or 864,060 trainable weights including the biases of the hidden layer, while a convolutional neural network with local receptive field of size 10 × 10 and 15 feature maps in the convolution layer will require (100 × 15 + 15) or 1,515 trainable weights, including the biases during computation.

Fig. 1 above shows a typical convolutional neural network; this paper has opted for the denotation of grouping corresponding convolution and pooling layers together as a layer, as can be seen in fig.1.

From Fig.1, it is assumed that the input image is of size K × L, a kernel of size a × a is used for convolution operation in C1 (first convolution layer), the number of feature maps is n, and the number of units in each feature map is i × i, as conceived in 2D. There are two approaches to shifting the kernels all over the image during convolution.

(a) *No padding approach*: In this approach, the kernel is shifted all over the image, without allowing the kernel to go outside the image borders. This means that some pixels that are close to the edges of the images will be left out during the convolution operation, depending on the size of the kernel used. Hence, the size of each resulting feature map is smaller than the input image.

(b) *Zero padding approach*: This approach allows the kernel to go over all pixels in the input, by padding with zeros, regions that fall outside the border of the input image during convolution. Hence, the whole input pixels can be convolved.

The relationship between the number of units in each feature map and the size of the kernel used in the convolution operation is given below.

$$M_x^L = \frac{M_x^{L-1} - K_x^L}{S_x^L + 1} + 1 \quad (1)$$

$$M_y^L = \frac{M_y^{L-1} - K_y^L}{S_y^L + 1} + 1 \quad (2)$$

Where, (M_x^L , M_y^L) is the feature map size of each plane, (K_x^L, K_y^l) is the kernel size shifted over the valid input image region, (S_x^L, S_y^L) is the skipping factor of kernels in x and y-directions between subsequent convolutions, and L indicates the layer. Each map in L_n is connected to at most M_{L-1} maps in layer L-1 [13].

The convolution operation outcome, g(i,j), for a 2D input function f(i,j), and kernel, v, also in 2D can be achieved using Equation 3.

$$g(i,j) = f(i,j) * v = \sum_{k=-n}^{n} \sum_{l=-n}^{n} v(k,l) f(i+k, j+l) \quad (3)$$

Copyright © 2016 MECS

The convolution, combination and implementation of feature maps can be achieved by using the relation in Equation 4.

$$x_j^l = f\left(\sum_{i \in M_j} x_i^{l-1} * k_{ij}^l + b_j^l\right) \quad (4)$$

Where j is the particular convolution feature map, M_j is a selection of input maps, k_{ij} is the convolution kernel, bj is the bias of each feature map, l is the layer in the network, and f is the activation function [14].

In layer 2 of the network (Fig.1), convolution operation is performed on sub-sampling layer S1, as we go deeper into the network, the number of feature maps implemented in each successive convolution layer increases. i.e. m > n. This compensates for the fact that spatial pooling operation reduces the dimensionality of the convolution feature maps each time. The particular sub-sampling feature maps that the convolution kernels will be convolved with in the layer preceding the classifier stage may be determined by building a convolution table, since there are now so many possible inputs (sub-sampling maps) to the convolution layer C3.

3.2 Pooling/Sub-sampling

Generally, this layer directly follows the convolution layer, its basic functions are to further reduce the dimensions of the feature maps and aggregate some features from the preceding layer. It is in this layer that the network develops some built-in invariance and covariance to features that are present in the input. Pooling is the primary source of dimension reduction and of local translation invariance in convolutional networks [15]. In pooling, a mask size determining what region of the preceding layer feature map is captured and operated on is chosen. There are as many sub-sampling maps as the convolution feature maps in any layer. i.e. each convolution feature map has its corresponding sub-sampling map or plane. The sub-sampling layer can be seen as S1 in Fig.1.

There are two common pooling operations, and are briefly described below.

- Average pooling: This method involves taking the average of the activation values of units in the preceding convolution feature map masked by the pooling window.
- Max pooling: This method involves taking the maximum value of the activation values of units in the preceding convolution feature map masked by the pooling window.

This research also aligns with the idea that the max pooling method aggregates features that are less sensitive to moderate invariance in the inputs; position invariance is achieved over larger local regions and the input image dimension is reduced along each direction [16]. Max-pooling leads to faster convergence rate by selecting superior invariant features which improve generalization performance [17].

Since overlap is usually not allowed in pooling operations, it therefore follows that dimension of sub-sampling feature maps is a fraction of the preceding convolutional feature map size by the pooling mask size. From Fig.1, it can be said that the size of sub-sampling maps in S1 is i/b × i/b, where i × i is the dimension or size of the preceding convolution feature maps and b × b is the size of the pooling window or mask. Furthermore, we can now infer that the number of units in each sub-sampling feature map can be obtained as i/b multiplied with i/b in 2D.

The last layer in convolutional neural networks is a regular classifier, which accepts the aggregated features of the preceding layer; the operation of the classifier is as obtains with any supervised learning classifier. For this research, a single hidden layer feedforward neural network is used in the classifier layer (module).

IV. Auto Encoder

An auto encoder is a generative neural network model; it can be used to explore underlying features that are present in data. These networks are 'grossly' feedforward networks, but in contrast, are not discriminative. They employ an unsupervised learning algorithm; hence, the fact that most data in real life are unlabelled can be leveraged on. In these networks, the input data also serve as the target data correspondingly; an auto encoder is required to learn the reconstruction of input data in the output layer. Thus, these networks build prior knowledge of features that do contribute to the successful reconstruction of the input data. i.e. the network is sensitive to redundant features in the input data. The auto encoder network can be seen as an encoder-decoder module; the encoder being the input-hidden layer interconnection, and the decoder, hidden-output layer interconnection [18].

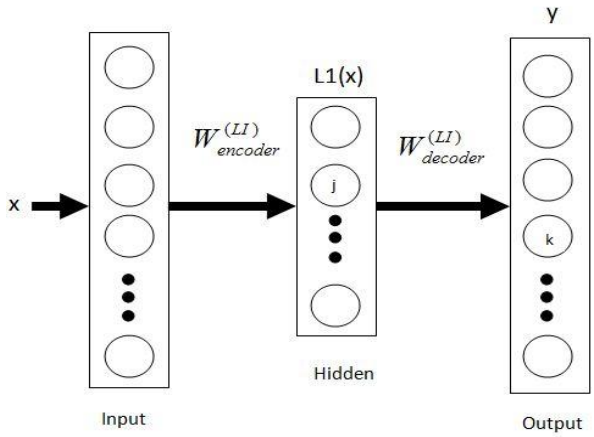


Fig.2. Auto encoder [18]

Fig. 2 shows an auto encoder, the equations relating the activations in the hidden and output layers are given below in Equations 5 & 6 [18].

Encoder:

$$L1(x) = g(m(x)) = sigm(b^{(L1)} + W_{encoder}^{(L1)} x) \quad (5)$$

Decoder:

$$y = z(n(x)) = sigm(b^{(y)} + W_{decoder}^{(L1)} L1(x)) \quad (6)$$

Where, m(x) and n(x) are the pre-activations of the hidden and output layers L1 and y respectively; $b^{(L1)}$ and $b^{(y)}$ are biases of the hidden and output layers, L1 and y respectively. Sometimes, $W_{encoder}^{(L1)}$ and $W_{decoder}^{(L1)}$ are tied using the relation $W_{decoder}^{(L1)} = (W_{encoder}^{(L1)})^T$, where T is the transpose function.

The network can be constrained, similar to what is achieved in the sparse coding approach, by making the number of neurons in the hidden layer smaller than the number of neurons in the output. i.e. the input features are compressed into the hidden layer, and thereafter expanded into the output layer again.

Learning is achieved by minimizing a cost function; for binary input variables, the sum of Bernoulli cross entropies is used as described by the equation 7.

$$C(x, y) = -\sum_{1}^{k} (x_k \log(y_k) + (1 - x_k) \log(1 - y_k)) \quad (7)$$

Where, x_k is the k-th binary variable of the input data, and y_k is the corresponding network output, k is the number of elements in the input and reconstructed output data. The MSE (Measn Squared Error) function can be used for real value inputs.

Auto encoders have significant usefulness in hybrid networks, where they are used as a pre-training technique in the initialization of feedforward network weights. It has been shown that the pre-training of multilayer networks initializes such networks in a weights space that is favourable for convergence to a better local minimum; this effect is important, and suffices even more in deep networks [19].

V. Convolutional Auto Encoder (CAE)

These networks leverage on the structure of conventional convolutional neural networks and auto encoders in overcoming the problems associated with either of the individual networks. i.e. as discussed in the section 2 and 3. Convolutional networks are well adapted to computer vision problem, while auto encoders are suited to optimization and regularization of deep networks through greedy layer-wise training. These two main features are combined in convolutional auto encoder.

In contrast to typical convolutional neural networks, where the weights are initialized randomly, a convolutional auto encoder network initializes it weights through an auto encoder. All features such as local connectivity, weight sharing, and pooling found in the conventional convolutional neural network remain valid in convolutional auto encoder networks. Hence, we can adapt some of the previous equations on convolutional neural networks and auto encoders to the learning of convolutional auto encoders using the tied weights approach.

$$L1(x)^n = g(m(x)) = sigm((b^{(L1)})^n + W_{encoder}^{(L1)} * x) \quad (8)$$

$$y^n = z(n(x)) = sigm((b^{(y)})^n + (W_{encoder}^{(L1)})^T * L1(x)) \quad (9)$$

Where, n denotes the n-th feature map in a convolution layer L1, and $W_{encoder}^{(L1)}$ is the kernel weight matrix, while $(b^{(L1)})^n$ and $(b^{(y)})^n$ are the biases of the n-th feature map for auto encoder convolution layer L_1 [20].

VI. Network Training and Testing

6.1 Data analysis

The two neural network architectures described above are trained on databases of Yoruba vowel characters. Hundreds of handwritten images of the characters are collected employing different people; they are then processed into the respective databases described below [18].

-Training database of Yoruba vowel characters: A1
-Validating database of Yoruba vowel characters: A2
-Translated database of Yoruba vowel characters: A3
-Rotated database of Yoruba vowel characters: A4
-Scale different database of Yoruba vowel characters: A5

Images in all the databases are processed as necessary. Networks are trained and validated on processed databases A1 and A2 respectively. Furthermore, we simulated or tested all the different trained networks with databases A3, A4, and A5 to obtain networks' performances on pattern invariance learning.

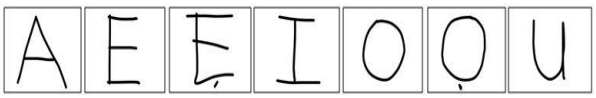


Fig.3. Unprocessed character images

Fig. 3 above shows the 7 unprocessed handwritten Yoruba vowel characters. These images are processed through the sequence: binarization of images, conversion to negatives, 10 × 10 median filtering, rescaling to 32 × 32 pixels using pattern averaging, and cropping of pattern occupied region of images. i.e. patterns centered. See Fig 4.

Fig.4. Processed characters

- Database A1:

The database samples used for the training of networks is shown in Fig. 5; the original handwritten characters are rotated in order to generate more samples for training.

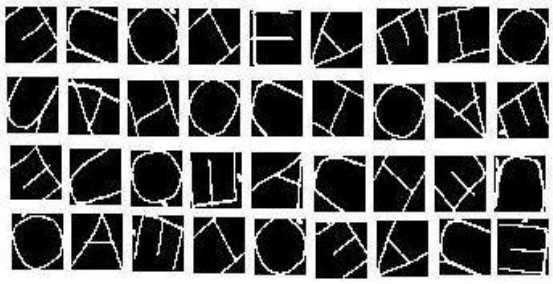


Fig.5. Training database characters

- Database A2:

This database is created to observe if over-fitting occurred during training of the networks. The database contains similar sample character images as in the training database A1. It is to be noted that none of the images found in this database is part of the training database A1. Samples of the characters in this database are shown in Fig.6.

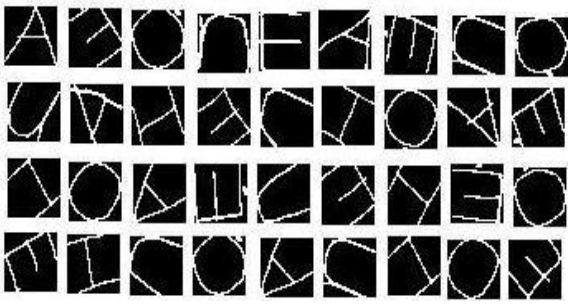


Fig.6. Validation database characters

- Database A3:

Fig. 7 show samples of the training and validation characters, but now translated spatially, vertically and more significantly horizontally in the images.

Fig.7. Translated characters database

This database is meant to test the performance of the trained networks on translation tolerance. i.e. built-in capability in recognizing translated versions of training samples. It is to be noted that the number of image pixels remains 32 × 32, as for all other databases within this work.

- Database A4:

In order to further investigate the built-in capability of the trained networks in recognizing rotated copies of the character samples, database A4, which is not part of the training and validation sets is used for testing. i.e. Fig.8.

Fig.8. Rotated database characters

- Database A5:

This database contains characters essentially in the training and validation databases, but now with various different scalings; the characters in this database have either been blown up or scaled down, as can be seen in Fig.9.

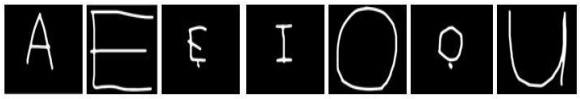


Fig.9. Scale variant database characters

6.2 CNN training

Input images are of size 32×32, and a kernel size or receptive field of 5×5 pixels is used in the first convolution layer (C1) to extract local features; 6 feature maps are extracted. The number of pixels for each feature map can be calculated as 28 × 28 using Equations 1 & 2 from section 3.1. A 2 × 2 sub-sampling mask with max pooling is used in the pooling layer of the first layer (S1). The same kernel size and sub-sampling mask as above are used for convolution maps (C2) and sub-sampling feature maps (S2) in the second layer. 12 convolution maps of size 5 × 5 are used in the layer preceding the classifier module. Many experiments were carried out to determine the suitable training parameters. A dual core, intel (R) Pentium (R) (2.00 GHz) CPU with 3GB RAM is used for all trainings and simulations. The final training parameters for the network are shown below in Table 1.

Table 1. Training parameters of CNN

Number of training samples	14,000
Activation function	Log-Sigmoid
Learning rate	0.65
Epochs	4,758
Training time (secs)	308
Mean Squared Error (MSE)	0.010

14,000 training samples are used, and after 4758 epochs, the network converged to a MSE of 0.010. The learning curve is shown in Fig. 10.

Also, a 10-fold cross-validation scheme is used to stop training. i.e. reducing the chances over-fitting.

Copyright © 2016 MECS

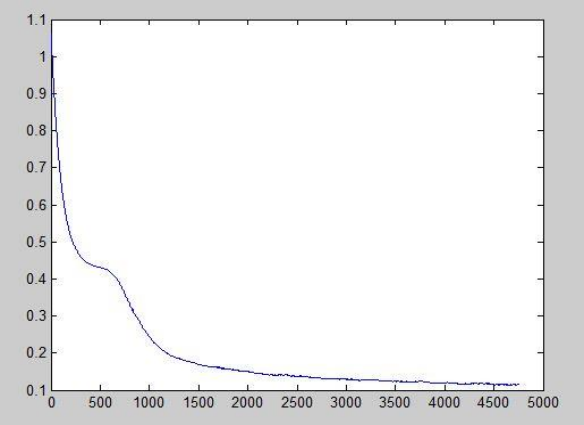


Fig.10. CNN learning curve

6.3 CAE training

The convolutional auto encoder network learnable weights are initialized as discussed in section 5, and the training parameters are shown below. A 5×5 filter was used for convolving the input and sub-sampling window of size 2×2 for pooling; 12 feature maps are extracted for the hidden layer.

Table 2. Training parameters of CAE

Number of training samples	14,000
Activation function	Log-Sigmoid
Learning rate	0.58
Epochs	1,189
Training time (secs)	158
Mean Squared Error (MSE)	0.008

The CAE is later fine-tuned discriminately for classification of the characters using the back propagation algorithm. The learning curve is shown in Fig 11. Also, a 10-fold cross-validation scheme is used to stop training.

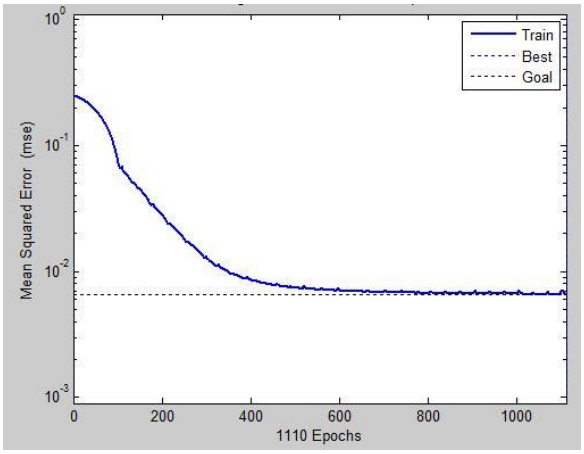


Fig.11. CAE learning curve

6.4 Network testing and discussion

The performances of the CNN and CAE are presented in this section. The networks are trained as described above using training database A1, validated using database A2; and then simulated with databases A3, A4, and A5, with each containing images with a particular invariance of interest as described in section 5. Table 3 summarizes the error rates achieved by the trained networks, including run times to simulate all databases for each network. Error rates (E.R) can be calculated using Equation 10.

$$E.R = \frac{No.of\ misclassified\ samples}{No.of\ test\ samples} \qquad (10)$$

Table 3. Error rates for network testing

Networks	Samples	CNN	CAE
Validation data: A2	2,500	2.78%	1.51%
Translated data: A3	700	67.14%	65.29%
Rotated data: A4	700	13.71%	12.71%
Scale varied data: A5	700	28.00%	26.57%
Run time (s)	4,600	8.67	5.58

For comparative analysis, we have compared the results obtained in this work with network performances achieved in an earlier published work on Yoruba character recognition using the same databases. The results obtained in the earlier work are given in Table 4 [18].

Table 4. Error rates for other networks [18]

Networks	Samples	BPNN	DAE	SDAE	DBN
Validation data	2,500	6.39%	6.79%	5.67%	3.77%
Translated data	700	82.86%	80.00%	74.29%	81.43%
Rotated data	700	27.29%	24.86%	22.14%	19.86%
Scale varied data	700	36.58%	30.57%	27.43%	23.29%

Where, BPNN is a back propagation neural network with 2 hidden layers, DAE is a denoising auto encoder, SDAE is a stacked denoising auto encoder with 2 hidden layers, and DBN is a deep belief network with 2 hidden layers.

It will be seen from table 3 & 4 that the CAE and CNN, on the average, outperform other networks in the earlier work, considering achieved error rates. Also, from Table 3, it can be seen that the CAE slightly outperforms the CNN on all the invariances considered; and has a lower average run time for testing compared to the CNN.

Since the training database, A1, contains rotated images, it follows that some prior knowledge about character rotation was built into the networks during the training phase of the networks due to the data manipulation just described above. Nevertheless, networks were still tested on rotational invariance using

Copyright © 2016 MECS

database A4. In contrast, all the networks have no prior knowledge on translation and scale invariance.

It can be explained that CNN and CAE, which implement convolution operations, achieved more invariance learning in comparison to BPNN, DAE, and SDAE, due to learning paradigms such as local receptive fields, weight sharing, and pooling operations.

In CNN, kernel weights are initialized randomly, hence, the weights may start out in a weight space that makes convergence less favourable. In contrast, the kernel weights in the CAE are learned by the auto encoder through an unsupervised pre-training scheme described in sections 3 and 4. The kernels learned in this manner have weights that are favourable to achieving both better optimization and regularization effects during training [19] [21], hence, the observed lower error rates obtained from the CAE network. Furthermore, it will be seen that the CAE took lesser training epochs and time for proper learning of the characters, compared to the CNN. Also, the run time for the CAE is lesser than the CNN. i.e. Table 3.

VII. Conclusion

This work investigates invariance learning in pattern recognition, an important constraint in many applications. We explore the achievable built-in invariance in neural networks which implement convolution operations, CNN (Convolutional Neural Network) and CAE (Convolutional Auto Encoder). Presented results show that CNN and CAE achieve better invariance learning compared to BPNN (Back Propagation Neural Network), DAE (Denoising Auto Encoder), SDAE (Stacked Denoising Auto Encoder) and DBN (Deep Belief Network).

Furthermore, we show that by leveraging on learned kernels through pre-training, better results can be obtained for the CAE as against the CNN.

It is to be noted that the classification task described in this work is quite hard considering that the characters to be recognized have diacritical marks, hence, an increase in the number of variations or achievable samples from each character; which when combined with the different writing styles of people, the problem may be seen as exponentially complex. The use of convolution based networks is more suited since the relative positions of diacritical marks can be captured during learning.

Lastly, it is the hope that advances in biological visual processing may suggest modifications or new architectures of neural networks that better lend themselves to intelligent recognition, and pattern invariance learning.

References

[1] L.G. Shapiro and G.C. Stockman: Computer Vision, Prentice-Hall, Inc., New Jersey, pp. 1-4. 2001.

[2] R.F. Abdel-Kader, R.M. Ramadan, F.W. Zaki: Rotation-Invariant Pattern Recognition Approach Using Extracted Descriptive Symmetrical Patterns, International Journal of Advanced Computer Science and Applications, vol. 3(5), pp. 151-158. 2012.

[3] Chen Guangyi, and Tien D. Bui: Invariant Fourier-wavelet descriptor for pattern recognition, Pattern Recognition, vol. 32(7), pp. 1083-1088. 1999.

[4] Prevost, Donald, et al.: Rotation, scale, and translation invariant pattern recognition using feature extraction, AeroSense 97, International Society for Optics and Photonics, pp. 255-264. 1997.

[5] S.K. Ranade, S. Anand: Empirical Analysis of Rotation Invariance in Moment Coefficients, International Journal of Computer Applications, vol. 119 (15), 2015.

[6] A. Khashman, B. Sekeroglu, K. Dimililer: Intelligent Rotation-Invariant Coin Identification System, WSEAS Transactions on Signal Processing, ISSN 1790-5022, Issue 5, vol. 2. 2006.

[7] J.Z. Leibo, et al.: Learning generic invariances in object recognition: translation and scale, Computer Science and Artificial Intelligence Laboratory Technical Report, pp. 1-6. 2010.

[8] D. Ciresan, U. Meier, & J. Schmidhuber: Multi-column deep neural networks for image classification, In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3642-3649. 2012.

[9] Wersing, Heiko, and Edgar Körner: Learning optimized features for hierarchical models of invariant object recognition, Neural computation vol. 15(7), pp. 1559-1588. 2003.

[10] C. Neubaurer.: Recognition of Handwritten Digits and Human Faces by Convolutional Neural Networks, International Computer Science Institute, 1996, pp. 1-9.

[11] K. Fukushima and T. Imagawa, Recognition and Segmentation of Connected Characters with Selective Attention, *Neural Networks* (6), 33-41. 1993.

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, pp. 6-10. 1998.

[13] D. C. Cires¸an, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber: Flexible, High Performance Convolutional Neural Networks for Image Classification, Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (2011), 1238-1243.

[14] J. Bouvrie, Notes on Convolutional Neural Networks, Center for Biological and Com-putational Learning, Massachusetts Institute of Technology, Cambridge, MA 02139, pp. 3-6. 2010.

[15] N. Sauder, Encoded Invariance in Convolutional Neural Networks, University of Chicago, pp. 2-6. 2006.

[16] D. Scherer, A. Muller, and S. Behnke, Evaluation of pooling operations in convolutional architectures for object recognition, Proc. of the Intl. Conf. on Artificial Neural Networks (2010), 92–101.

[17] J. Nagi, F. Ducatelle, G.A. Di Caro, at el.: Max-Pooling Convolutional Neural Networks for Vision-based Hand Gesture Recognition, IEEE International Conference on Signal and Image Processing Applications (2011), 343-349.

[18] O.K. Oyedotun, E.O. Olaniyi, and A. Khashman, Deep Learning in Character Recognition considering Pattern Invariance Constraints, *International Journal of Intelligent Systems and Applications*, 7 (7), pp. 1-10. 2015. DOI: 10.5815/ijisa.2015.07.01.

[19] L. Deng, An Overview of Deep-Structured Learning for Information Processing, Asia-Pacific Signal and Information Processing Association: Annual Summit and Conference (2014), 2-4.

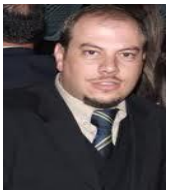
Copyright © 2016 MECS

[20] J. Masci, U. Meier, D. Cire şan, and J. Schmidhuber.: Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction: ICANN 2011, T. Honkela et al., ed., Part I, Springer-Verlag Berlin Heidelberg, LNCS 6791, pp. 52–59. 2011.

[21] D. Erhan at el.: Why Does Unsupervised Pre-training Help Deep Learning?,Appearing in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS), Chia Laguna Resort, Sardinia, Italy (2010): Vol. 9 of *JMLR*: W&CP 9, 1-7.

Authors' Profiles

Oyebade K. Oyedotun in 2015 received M.Sc. degree in Electrical & Electronic Engineering (major: machine learning) from Near East University, Lefkosa, Turkey. From 2013-2014 he was a member of the Intelligent Systems Research Group (ISRG) at Near East University, Lefkosa, Turkey. Between 2014-2015, he was a member of the Centre of Innovation for Artificial Intelligence (CiAi) at British University of Nicosia, in Girne, Turkey. Since 2015, he is a senior member of the European Centre for Research and Academic Affairs (ECRAA) in Lefkosa, Turkey. Research interests include artificial neural networks, pattern recognition, machine learning, vision systems, fuzzy systems, swarm intelligence, cognition modeling and robotics. PH: +905428892591. E-mail: oyebade.oyedotun@ecraa.com, oyebade.oyedotun.k@ieee.org.



Kamil Dimililer received B.Sc. and MSc. degrees in Electrical and Electronic Engineering from Near East University, N. Cyprus. After a short period of time, he started his PhD. and he received the PhD. Degree in the same field in 2014. He has been assigned as Deputy Chairman in Automotive Engineering for a short period of time and currently he is an Assistant Professor in the Department of Automotive Engineering as Chairman in 2015. Assist. Prof. Dr. Kamil Dimililer has 9 International Journal Publication, 16 Conference Publications, 3 Book Chapters in International Books and 7 publications in National Journals. His research interests include image processing, pattern recognition and intelligent systems. E-mail: kamil.dimililer@neu.edu.tr

How to cite this paper: Oyebade K. Oyedotun, Kamil Dimililer,"Pattern Recognition: Invariance Learning in Convolutional Auto Encoder Network", International Journal of Image, Graphics and Signal Processing(IJIGSP), Vol.8, No.3, pp.19-27, 2016.DOI: 10.5815/ijigsp.2016.03.03