Automatic Image Annotation Using Convex Deep Learning Models

Niharjyoti Sarangi and C. Chandra Sekhar
Department of CSE, Indian Institute of Technology Madras, Chennai, India

Keywords: Image Annotation, Tensor Deep Stacking Networks, Kernel Deep Convex Networks, Deep Convolutional Network, Deep Learning.

Abstract: Automatically assigning semantically relevant tags to an image is an important task in machine learning. Many algorithms have been proposed to annotate images based on features such as color, texture, and shape. Success of these algorithms is dependent on carefully handcrafted features. Deep learning models are widely used to learn abstract, high level representations from raw data. Deep belief networks are the most commonly used deep learning models formed by pre-training the individual Restricted Boltzmann Machines in a layer-wise fashion and then stacking together and training them using error back-propagation. In the deep convolutional networks, convolution operation is used to extract features from different sub-regions of the images to learn better representations. To reduce the time taken for training, models that use convex optimization and kernel trick have been proposed. In this paper we explore two such models, Tensor Deep Stacking Network and Kernel Deep Convex Network, for the task of automatic image annotation. We use a deep convolutional network to extract high level features from raw images, and then use them as inputs to the convex deep learning models. Performance of the proposed approach is evaluated on benchmark image datasets.

1 INTRODUCTION

Developing techniques for efficient extraction of usable and meaningful information has become increasingly important with the explosive growth of digital technologies. Low level features like color, texture and shape can be used to classify images into different categories. However, in many cases it is not suitable to use a single class label because of the presence of more than one semantic concept in an image. One way to handle this is by assigning multiple relevant keywords to a given image, reflecting its semantic content. This is often referred to as image annotation.

Learning techniques such as Binary Relevance (Boutell et al., 2004) and Classifier Chains (Read et al., 2011), transform an annotation task into a task of binary classification. Another approach to tackle the problem of annotation is by adapting popular learning techniques to deal with multiple labels directly (Tsoumakas and Katakis, 2007; Zhang and Zhou, 2014). Multi Label k-Nearest Neighbors (ML-kNN) (Zhang and Zhou, 2007), Multi Label Decision Tree (ML-DT) (Vens et al., 2008) and Rank-SVM (Elisseeff and Weston, 2001) are some of the commonly used methods in this category. Rank-SVM is a ranking based approach coupled with a set size predictor which uses Support Vector Machines to minimize the ranking loss while having a large margin. Among other models, semantic space auto-annotation model (Hare et al., 2008) constructs a special form of a vector space, called a semantic space, from the labels associated with the images. Images are projected into this space in order to be retrieved or annotated. Latent semantic analysis (Hofmann, 1999) is used to build this space. The success of these techniques is largely dependent on the effectiveness of the features used.

Learning representations of the data that makes it easier to extract useful information is highly desirable (Bengio et al., 2013) for developing a good classification or annotation framework. Deep learning models are the commonly used techniques for learning representation from raw data. These models aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features, as illustrated in Figure 1.

Deep learning models such as Deep Belief Networks (DBN) (Hinton and Salakhutdinov, 2006) and Deep Boltzmann Machines (DBM) (Salakhutdinov and Hinton, 2009; Ranzato et al., 2010; Montavon et al., 2012) have performed well in classification and

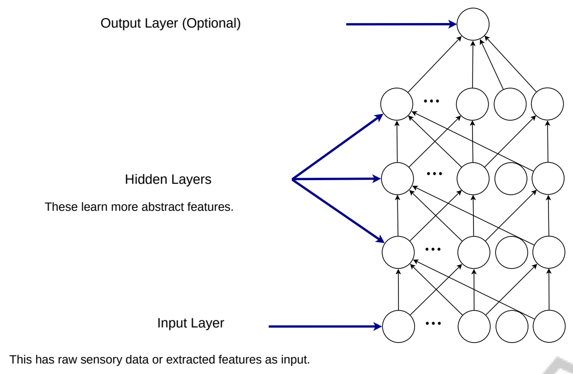
Output Layer (Optional)
Hidden Layers
These learn more abstract features.
Input Layer
This has raw sensory data or extracted features as input.

Figure 1: Scheme of learning representations in a multilayered network. Raw pixel values or extracted features are given as input. The input layer is followed by multiple hidden layers that learn increasingly abstract representation.

recognition tasks (Le Roux and Bengio, 2008). These models are formed by pre-training individual layers (Hinton et al., 2006a) and then stacking together and training them using error back-propagation. Each layer of a DBN consists of an energy-based model known as Restricted Boltzmann Machine (RBM). An RBM is trained using contrastive divergence to obtain a good reconstruction of the input data (Hinton et al., 2006b). Contrastive divergence and error back-propagation are computationally complex methods. In Deep Convolutional Networks (LeCun et al., 2010; Lee et al., 2009; Krizhevsky et al., 2012), convolution operation is used to extract features from different sub-regions of an image to learn a better representation. Although Deep Convolutional Networks are trained completely using error back-propagation, they use sub-sampling layers to reduce the number of inputs to each layer. To solve the issue of complexity, a model known as Deep Stacking Network (DSN) (Deng et al., 2012b) that consists of many stacking modules was recently proposed. Each module is a specialized neural network consisting of a single non-linear hidden layer and linear input and output layers. Since convex optimization is used to speedup the learning in each module, this model is also called as Deep Convex Network (DCN) (Deng and Yu, 2011). Tensor Deep Stacking Networks (T-DSN) (Hutchinson et al., 2013) , introduced as an extension of the DSN architecture, captures better representations by using two sets of nonlinear nodes in the hidden layer. The T-DSN model has been shown to perform better than the DSN model for image classification and phone recognition tasks. Kernel Deep Convex Network(K-DCN) (Deng et al., 2012a) on the other hand uses kernel trick so that the number of hidden nodes in each module is unbounded.

In this paper, we propose a framework that uses convex deep learning models (T-DSN and K-DCN) for the task of image annotation. We also propose using the features extracted from a Deep Convolutional Network as input to the convex models. The remainder of this paper is organized as follows: Section 2 gives a brief discussion on T-DSN and K-DCN. In Section 3 we describe the details of our experiments and compare the results with the existing methods.

2 CONVEX DEEP LEARNING MODELS

2.1 Tensor Deep Stacking Networks

A tensor deep stacking network is a generalized form of a deep stacking network. The input data is provided to the nodes in the input layer of the first module. The input to the higher modules is obtained by appending output from the module just below it to the original input data. Unlike DSN, each module of TDSN has two sets of hidden layer nodes and thus, two sets of connections between the input layer and the hidden layer as shown in Figure 2. The output layer nodes are bilinearly dependent on the hidden layer nodes.

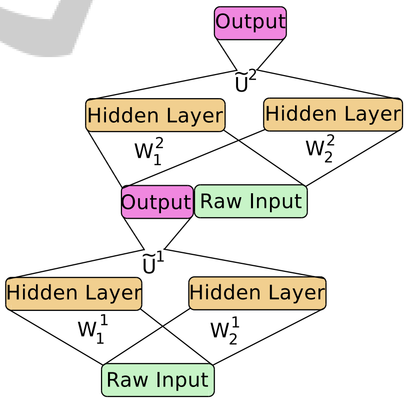

Figure 2: Architecture of tensor deep stacking network.

Let the target vectors **t** be arranged to form the columns of matrix T, the input data vectors **v** be arranged to form the columns of matrix V, and H_1 and H_2 denote the set of matrices of the outputs of the hidden units. There are two sets of lower weight parameters (W_1 and W_2). They are associated with connections from the input layer to the two hidden layers containing L_1 and L_2 sigmoidal nodes respectively.

Since the hidden layers contain sigmoidal nodes, the output of a hidden layer can be expressed as:

$$\begin{aligned} H_1 &= logistic(W_1^T V) \\ H_2 &= logistic(W_2^T V) \end{aligned} \tag{1}$$

Let $\mathbf{h}_1$ be the vector of outputs from the first set of hidden nodes and $\mathbf{h}_2$ be the vector of outputs from the second set of hidden nodes. Let h_{1i} be the i^{th} entry in $\mathbf{h}_1$ and h_{2j} be the j^{th} entry in $\mathbf{h}_2$.

If C is the number of nodes in the output layer, weights of connections from hidden layers to the output layer are represented as a tensor $\mathbf{U} \in \mathbf{R}^{L_1} \times \mathbf{R}^{L_2} \times \mathbf{R}^{C}$. The tensor $\mathbf{U}$ can be considered as a 3-dimensional matrix.

Let y_k denote the output of k^{th} node in output layer. The output vector can be obtained by computing $(\mathbf{U} \times_1 \mathbf{h}_1) \times_2 \mathbf{h}_2$ where $\times_i$ stands for multiplication along the i^{th} dimension. In a simplified notation

$$y_k = \sum_{i=1}^{L_1} \sum_{j=1}^{L_2} U_{ijk} h_{1i} h_{2j} \tag{2}$$

Let

$$\tilde{\mathbf{h}} = \mathbf{h}_1 \otimes \mathbf{h}_2$$

where $\otimes$ is the Kronecker product. Let $\tilde{\mathbf{u}}_k$ be the vectorized version of matrix U_k in which all columns are appended to form a single vector. The matrix U_k is obtained by setting the third dimension of tensor $\mathbf{U}$ equal to k. Hence, length of $\tilde{\mathbf{u}}_k$ is $L_1 L_2$. Now, we can rewrite equation (2) as,

$$y_k = \tilde{\mathbf{u}}_k{}^T \tilde{\mathbf{h}} \tag{3}$$

Arranging all $\tilde{\mathbf{u}}_k$'s for $k = 1, 2, ..., C$, into a matrix $\tilde{U}$ = [$\tilde{\mathbf{u}}_1$ $\tilde{\mathbf{u}}_2$ ... $\tilde{\mathbf{u}}_C$], the overall prediction becomes

$$\mathbf{y} = \tilde{U}^T \tilde{\mathbf{h}} \tag{4}$$

where $\mathbf{y}$ is the estimate of target vector $\mathbf{t}$.

Thus, bilinear mapping from two hidden layers can be seen as a linear mapping from an implicit hidden representation $\tilde{\mathbf{h}}$. Aggregating the implicit hidden layer representations for each of the N instances into the columns of an $L_1 L_2 \times N$ matrix $\tilde{H}$, we obtain

$$Y = \tilde{U}^T \tilde{H} \tag{5}$$

where $\tilde{H}$ contains $\mathbf{h}_k$ in k^{th} column.

The convex formulation for $\tilde{U}$ in this case is,

$$min_{\tilde{U}^T} \|\tilde{U}^T H - T\|^2 \tag{6}$$

where $\|.\|^2$ represents the squared norm operation.

Solving the optimization (6) we get:

$$\tilde{U}^T = T\tilde{H}^T (\tilde{H}\tilde{H}^T)^{-1} \tag{7}$$

We see that the output of each hidden node in first layer appears L_2 number of times in $\tilde{\mathbf{h}}$. So, we have to add errors due to all those terms in order to get the error caused by this particular node. Hence, the equation for weight update needs to be modified to account for this and the modified equations are:

$$\Delta W_1 = \eta V[H_1^T \circ (\Gamma - H_1^T) \circ \Psi_1] \tag{8}$$

$$\Delta W_2 = \eta V[H_2^T \circ (\Gamma - H_2^T) \circ \Psi_2] \tag{9}$$

Here $\circ$ is the element-wise multiplication of two matrices, Γ is a matrix of all ones, η is the learning rate and

$$\begin{aligned} \Psi_{1nk} &= \sum_{k=1}^{L_2} H_{2nk} \tilde{\Theta}_{((i-1)L_2+k),n} \\ \Psi_{2nk} &= \sum_{k=1}^{L_1} H_{1nk} \tilde{\Theta}_{((i-1)L_1+k),n} \end{aligned} \tag{10}$$

$$\tilde{\Theta} = 2\tilde{H}^{+}(\tilde{H}T^T)(T\tilde{H}^{+}) - 2T^T(T\tilde{H}^{+}) \tag{11}$$

Here H_1 is the matrix of outputs of nodes in the first hidden layer, H_2 is the matrix of outputs of nodes in the second hidden layer. The dimensions of matrices Ψ_1 and Ψ_2 are $N \times L_1$ and $N \times L_2$ respectively. Each of these two matrices Ψ_1 and Ψ_2 acts as a bridge between high dimensional implicit representation $\tilde{\mathbf{h}}$ and low dimensional representations $\mathbf{u}$ and $\mathbf{v}$.

Since T-DSN uses convex optimization techniques to directly determine the upper-layer weights, the training time is greatly reduced. However, computing the lower-layer weights is still an iterative process.

2.2 Kernel Deep Convex Networks

A kernel deep convex network (K-DCN), like a T-DSN, is composed by stacking of shallow neural network modules. This model completely eliminates the non-convex learning for the lower-layer weights using the kernel trick. In case of K-DCN, a regularization term C is included in the expression for computing the upper-layer weights U. This modification helps bound the values of elements of U and prevents the model from over-fitting on the training data.

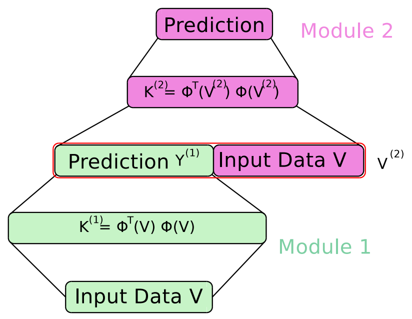

Figure 3: Architecture of kernel deep convex network with two modules.

The formulation for U takes the form of,

$$min_U[\frac{1}{2} * Tr\{(Y-T)^T(Y-T)\} + \frac{C}{2}U^T U]$$

where Y is the predicted output for the output nodes and T is the target output. The closed form expression for U is obtained by solving this minimization as follows :

$$U = (CI + HH^T)^{-1} HT^T \tag{12}$$

The output of the given module of KDCN is given by,

$$\mathbf{y}_k = TH^T(CI + HH^T)^{-1}\mathbf{h}_i \tag{13}$$

The sigmoidal function of hidden units is replaced with a generic nonlinear mapping function $\Phi(\mathbf{v})$ from the raw input features $\mathbf{v}$. The mapping $\Phi(\mathbf{v})$ will have high-dimensionality (possibly infinite) which is determined implicitly by a chosen kernel function. The unconstrained optimization problem can be reformulated as follows :

$$min_U[\frac{1}{2} * Tr\{(Y-T)^T(Y-T)\} + \frac{C}{2}U^TU]$$

subject to

$$T - U^TG(V) = Y - T$$

where columns of $G(V)$ are formed by applying the transformation $\Phi(.)$ on each input $\mathbf{v}$. Solving this problem gives

$$U = G(V)(CI + K)^{-1}T^T \tag{14}$$

where $K = G^T(V)G(V)$ is the kernel gram matrix of V.

Finally, for each new input vector $\mathbf{v}$ in the test set, the prediction of KDCN module is given by

$$\mathbf{y}(\mathbf{v}) = U^T\Phi(\mathbf{v}) = T(CI + K)^{-1}\mathbf{k}^T(\mathbf{v}) \tag{15}$$

Here $\mathbf{k}(\mathbf{v})$ is the kernel vector such that $k_n(\mathbf{v}) = k(\mathbf{v}_n, \mathbf{v})$ and $\mathbf{v}_n$ is a vector from training set.

For the subsequent modules, the output of nodes in the output layer is appended with the raw input. For l^{th} module ($l > 2$) equation (14) is valid with a slight modification in the kernel function to account for this extra input as follows :

$$K = G^T(Z)G(Z) \tag{16}$$

where $Z = V|Y^{(l-1)}|Y^{(l-2)}|....|Y^1$, Y^m is the prediction of module m, and $U|V$ represents the concatenation of U and V.

Using the equations (15) and (16) we eliminate the need of back-propagation and get a convex expression for training the model. The KDCN model combines the power of deep learning and kernel learning in a principled way. It is fast because there is no back-propagation.

2.3 Framework for Image Annotation

If a concept is present in an image, the corresponding bit in a binary target output vector **t** is turned on. Each module of a T-DSN is trained to predict **t**. Once the module is trained and the weights W_1, W_2, and U are learned, equation (4) is used to compute the estimated output. For the higher modules, the input data is concatenated with the output of the module below it (or with the output of n modules below it) and used as an augmented input. This process is repeated for all the modules and the output obtained at the last module is retained. Similarly, in case of a K-DCN, equation (15) is used to find predictions for each module.

One of the following methods to obtain the annotation labels from the outputs of a model is used.

1. A threshold value is decided empirically using a held-out validation set. In the estimated output vectors, if the posterior probability value for a particular concept exceeds the threshold, it is considered as an annotation label for the image.
2. Based on the average number of labels present in the images, a value k is selected. An image is annotated with those concepts that correspond to the top k values in the estimated output vector.

3 EXPERIMENTS AND RESULTS

In this section, we present the details of image annotation datasets used and the experimental results for T-DSN and K-DCN. We compare the performance of these models with the state-of-the-art performance.

3.1 Experimental Setup

We used MATLAB on an Intel i7 8-core CPU with 16 GB of RAM for running the Rank-SVM. For T-DSN and K-DCN, we used NVIDIA Tesla K20C GPU with CUDA.

In order to reduce the number of multiplications in the computation of $\tilde{\Theta}$, equation (11) is re-written as:

$$\begin{aligned}\tilde{\Theta} &= 2(\tilde{H}^+\tilde{H}T^T - T^T)(T\tilde{H}^+) \\ &= 2(\tilde{H}^+\tilde{H}T^T - T^T)\tilde{U}^+\end{aligned} \tag{17}$$

In order to reduce the memory requirements for the computation of $\tilde{\Theta}$, equation (17) is parenthesized as follows:

$$\tilde{\Theta} = 2(\tilde{H}^+(\tilde{H}T^T) - T^T)\tilde{U}^+ \tag{18}$$

In this order of multiplication, we avoid computing $\tilde{H}^+\tilde{H}$, which is a $N \times N$ matrix. In general, the value of N is large (20,000 - 50,000). Accommodating such a large matrix in the GPU memory is problematic. Many matrices are reused in the process of training. Matrices are allocated memory only when required and freed immediately after their use in order to make the best use of memory available.

For K-DCN, we used three different types of kernel functions, namely, Gaussian kernel, Polynomial kernel and Histogram Intersection Kernel (HIK). The kernel parameters and regularization parameter were tuned to obtain a range of values for the first module. For the later modules, the tuning is done with respect to the range of parameters obtained for the previous module, and a set of globally optimum parameters was obtained.

3.2 Feature Extraction

We used a deep convolutional network to obtain a useful representation from an image. A deep convolutional network consists of several layers. A convolutional layer consists of a rectangular grid of neurons. Each neuron takes inputs from a rectangular section of the previous layer. The weights for this rectangular section are constrained to be the same for each neuron in the convolutional layer. Constraining the weights makes it work like many different copies of the same feature detector applied to different positions. This constraint also helps in restricting the number of parameters. The output of a neuron in the convolutional layer, l for a filter of size $(m*n)$ is given by

$$s_{ij}^{l} = f\left(\sum_{x=0}^{m}\sum_{y=0}^{n} w_{xy} s_{(x+i)(y+j)}^{(l-1)}\right) \qquad (19)$$

where $f(x) = \log(1+e^x)$. This nonlinearity was approximated using a simpler function, $f(x) = \max(0,x)$, which is known as the rectifier function. The nodes that use the rectifier function are referred to as Rectified Linear Units (ReLU). Use of ReLU reduced the time taken significantly.

The pooling layer takes outputs of small rectangular blocks in the convolutional layer and subsamples it to produce a single output from that block. The pooling layer can take the average, or maximum, or learn a linear combination of outputs of the neurons in the block. In all our experiments, we used max-pooling. Pooling helps the network achieve small amount of translational invariance at each level. Also, it reduces the number of inputs to the next layer. Finally, after two convolutional and max-pooling layers, we added two fully connected layers. The activity of the nodes in the last fully connected layer was used as input to the T-DSN and K-DCN models.

Apart from this, we also used the SIFT features (Lowe, 2004) as input to the deep learning models.

3.3 Datasets Used

We test our models with two real-world datasets that contain color images with their annotations: University of Washington annotation benchmark dataset (Washington, 2004) and the MIRFLICKR-25000 collection (Huiskes and Lew, 2008).

The Washington dataset had 1109 color images corresponding to 22 different categories with an average annotation length of 6. Out of all the concepts available, we selected only 45 concepts that had more than 25 images associated with each of them. The list of these 45 concepts is given in Table 1.

Table 1: List of 45 concepts selected for our study on University of Washington annotation benchmark dataset.

trees	bushes	grass	sidewalk	ground
rock	flowers	camp	sky	trees
trunk	people	water	dog	woman
street	cars	pole	house	beach
ocean	clouds	mountain	river	building
lantern	window	bridge	band	man
stone	Snow	Boats	sun	Huskies
football	Stadium	stand	field	hiker
mosque	frozen	players	temple	smoke

Some of the images from this dataset with their annotation labels are shown in Figure 4. Because of the small number of images, we do not use convolutional features for this dataset.

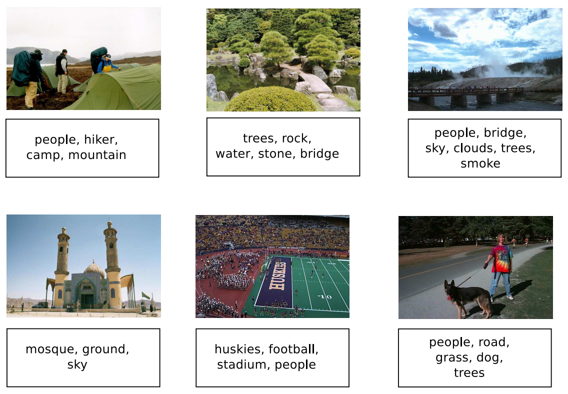
people, hiker, camp, mountain
trees, rock, water, stone, bridge
people, bridge, sky, clouds, trees, smoke
mosque, ground, sky
huskies, football, stadium, people
people, road, grass, dog, trees

Figure 4: Illustration of images with their annotation labels from the University of Washington annotation benchmark dataset.

MIRFLICKR-25000 is a database of 25,000 color images belonging to various categories. The average number of tags per image is 9. Some of the images from this dataset with their annotation labels are shown in Figure 5. For our studies, we consider the

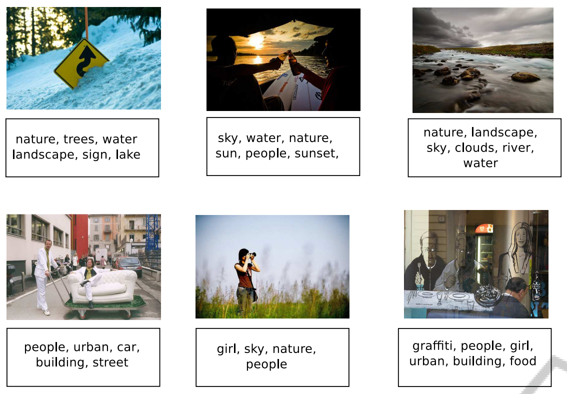

Figure 5: Illustration of images with their annotation labels from the MIRFLICKR dataset.

30 most frequently occurring tags. These tags have at least 150 images associated with each of them.

We randomly selected 30% of the images for testing, and repeated our studies over 5 folds.

3.4 Results

A T-DSN consisting of 3 modules with 100 nodes in each of the hidden layers was used on the University of Washington dataset. In our experiments we observed that having the same number of nodes in both the sets of hidden nodes generally give a better performance.

The precision, recall and F-measure for different thresholds in the threshold based decision logic are reported in Table 2.

We repeated the previous experiment with different values of *k* in the top-*k* based decision logic, and the precision, recall, and F-measure values are reported in Table 3.

We repeated these experiments with K-DCN. Best performance was observed for a Gaussian Kernel. The results of these experiments are reported in Table 4 and Table 5.

It is observed that the F-measure values for K-DCN are slightly lower when compared with that for T-DSN. One of the possible reasons for this could be that the kernel parameters used might not be the best. The state-of-the-art methods for image annotation, namely, Rank-SVM and semantic space model give F-measure values of 0.61 and 0.63 respectively. Figure 6 compares the actual annotation labels for some randomly selected images in University of Washington dataset with the annotations generated by the T-DSN model.

It is observed that the number of annotation labels generated by the models were slightly higher than that of the ground truth. In many cases, the extra labels are somehow related to the image.

Table 2: Precision, recall, and F-measure for different thresholds in the threshold based decision logic for annotation of images in the University of Washington data with T-DSN.

Threshold	Precision	Recall	F-measure
0.20	0.50	0.87	0.64
0.25	0.58	0.81	0.67
0.30	0.63	0.75	0.68
0.35	0.68	0.70	**0.68**
0.40	0.72	0.64	0.67
0.45	0.73	0.58	0.64
0.50	0.75	0.47	0.58

Table 3: Precision, recall, and F-measure for different values of *k* in the top-*k* based decision logic for annotation of images in the University of Washington data with T-DSN.

k	Precision	Recall	F-measure
2	0.42	0.27	0.33
3	0.51	0.46	0.48
4	0.52	0.60	0.56
5	0.50	0.70	0.58
6	0.49	0.79	0.60
7	0.44	0.84	0.58
8	0.41	0.88	0.56

Table 4: Precision, recall, and F-measure for different thresholds in the threshold based decision logic for annotation of images in the University of Washington data with K-DCN.

Threshold	Precision	Recall	F-measure
0.20	0.39	0.90	0.54
0.25	0.50	0.84	0.63
0.30	0.52	0.81	0.63
0.35	0.59	0.76	0.66
0.40	0.64	0.68	0.66
0.45	0.73	0.59	0.65
0.50	0.77	0.49	0.59

Table 5: Precision, recall, and F-measure for different values of *k* in the top-*k* based decision logic for annotation of images in the University of Washington data with K-DCN.

k	Precision	Recall	F-measure
2	0.31	0.21	0.25
3	0.43	0.49	0.46
4	0.48	0.53	0.50
5	0.48	0.61	0.54
6	0.46	0.68	0.55
7	0.41	0.77	0.53
8	0.37	0.85	0.52

For the MIRFLICKR dataset, the study is carried out using the SIFT features and convolutional fea-

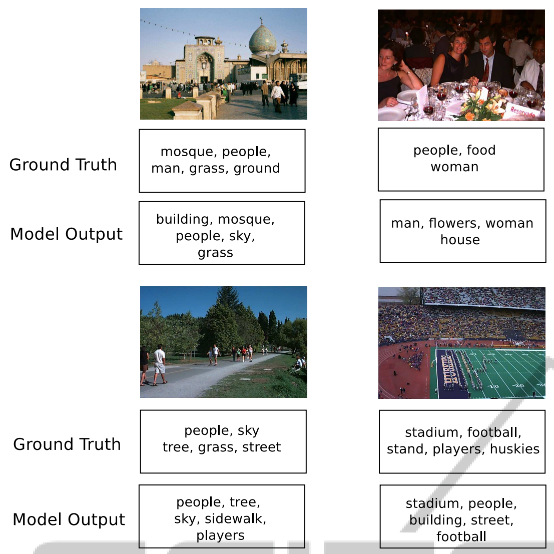
Ground Truth
mosque, people, man, grass, ground
people, food woman
Model Output
building, mosque, people, sky, grass
man, flowers, woman house
Ground Truth
people, sky tree, grass, street
stadium, football, stand, players, huskies
Model Output
people, tree, sky, sidewalk, players
stadium, people, building, street, football

Figure 6: Illustration of images with actual annotation labels and predicted annotation labels in the University of Washington dataset with T-DSN.

Table 6: Performance comparison of models for image annotation task on MIRFLICKR dataset.

Model	**Input Features**	**F-Measure**
Semantic Spae	SIFT	0.26
Rank-SVM	SIFT	0.25
TDSN	SIFT	0.24
TDSN	Convolutional	0.29
KDCN	SIFT	0.26
KDCN	Convolutional	**0.34**

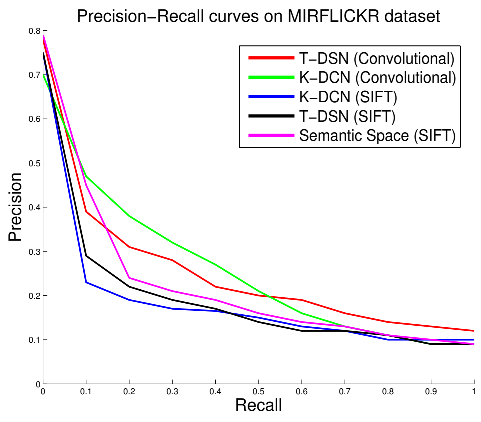

Figure 7: Precision-recall curves for different models on MIRFLICKR dataset.

tures. Fig. 7 shows the precision-recall curves for different models. The best F-measure values for different models are presented in Table 6.

It is observed that K-DCN and T-DSN perform better with convolutional features. It is also noted that convex deep learning methods perform better than the semantic space annotation method.

4 SUMMARY AND CONCLUSIONS

In this paper, we used the convex deep learning models, such as T-DSN and K-DCN for image annotation tasks. We also used features extracted from a deep convolutional network for this task. Through the experimental studies, it is observed that the T-DSN and K-DCN models with convolutional features as input give an improved performance. Once the convolutional network is trained on a large set of images, it is easy to extract features. The convex networks take less time to train, making them useful for image annotation tasks in practice.

For the K-DCN model, we have used only a single kernel function for a module. We can extend this by using multiple types of kernel functions. Finding a set of globally optimal parameters for K-DCN is difficult. Similarly, for T-DSN we observed that having different number of nodes in each hidden layer is not beneficial. However, we did not find any criterion for selecting the suitable number of hidden layer nodes. A recipe for selecting the number of nodes in T-DSN and globally optimum parameters for K-DCN will be useful.

REFERENCES

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. (2004). Learning multi-label scene classification. *Pattern Recognition*, 37(9):1757–1771.

Deng, L., Tür, G., He, X., and Hakkani-Tür, D. Z. (2012a). Use of kernel deep convex networks and end-to-end learning for spoken language understanding. In *IEEE Workshop on Spoken Language Technologies*, pages 210–215.

Deng, L. and Yu, D. (2011). Deep convex network: A scalable architecture for speech pattern classification. In *Interspeech*.

Deng, L., Yu, D., and Platt, J. (2012b). Scalable stacking and learning for building deep architectures. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*.

Elisseeff, A. and Weston, J. (2001). A kernel method for multi-labelled classification. In *Advances in Neural Information Processing Systems 14*, pages 681–687.

Hare, J., Samangooei, S., Lewis, P., and Nixon, M. (2008). Semantic spaces revisited: investigating the performance of auto-annotation and semantic retrieval using semantic spaces. In *Proceedings of the International conference on Content-based image and video retrieval*, pages 359–368.

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

Hinton, G. E., Osindero, S., Welling, M., and Teh, Y. W. (2006b). Unsupervised discovery of nonlinear structure using contrastive backpropagation. *Cognitive Science*, 30(4):725–731.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Hofmann, T. (1999). Probabilistic latent semantic analysis. In *Proceedings of the Uncertainty in Artificial Intelligence*, pages 289–296.

Huiskes, M. J. and Lew, M. S. (2008). The mir flickr retrieval evaluation. In *Proceedings of the 2008 ACM International Conference on Multimedia Information Retrieval*.

Hutchinson, B., Deng, L., and Yu, D. (2013). Tensor deep stacking networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1944–1957.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the Neural Information Processing System*, volume 22, pages 1106–1114.

Le Roux, N. and Bengio, Y. (2008). Representational power of restricted Boltzmann machines and deep belief networks. *Neural Computation*, 20(6):1631–1649.

LeCun, Y., Kavukcuoglu, K., and Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of International Symposium on Circuits and Systems*, pages 253–256.

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). 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.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110.

Montavon, G., Braun, M. L., and Mller, K.-R. (2012). Deep Boltzmann machines as feed-forward hierarchies. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 22, pages 798–804.

Ranzato, M., Krizhevsky, A., and Hinton, G. E. (2010). Factored 3-way restricted Boltzmann machines for modeling natural images. *Journal of Machine Learning Research - Proceedings Track*, 9:621–628.

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359.

Salakhutdinov, R. and Hinton, G. (2009). Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pages 448–455.

Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13.

Vens, C., Struyf, J., Schietgat, L., Džeroski, S., and Blockeel, H. (2008). Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214.

Washington, U. (2004). Washington ground truth database. http://www.cs.washington.edu/research/imagedatabase.

Zhang, M.-L. and Zhou, Z.-H. (2007). Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038 – 2048.

Zhang, M.-L. and Zhou, Z.-H. (2014). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837.