QUARTERLY OF APPLIED MATHEMATICS
VOLUME LXXVII, NUMBER 2
JUNE 2019, PAGES 207–226
https://doi.org/10.1090/qam/1525
Article electronically published on January 18, 2019

DEFORMABLE CLASSIFIERS

By

JIAJUN SHEN (*Department of Computer Science, University of Chicago, Chicago, Illinois 60637*)

AND

YALI AMIT (*Department of Statistics, University of Chicago, Chicago, Illinois 60637*)

Abstract. Geometric variations of objects, which do not modify the object class, pose a major challenge for object recognition. These variations could be rigid as well as non-rigid transformations. In this paper, we design a framework for training deformable classifiers, where latent transformation variables are introduced, and a transformation of the object image to a reference instantiation is computed in terms of the classifier output, *separately for each class.* The classifier outputs for each class, after transformation, are compared to yield the final decision. As a by-product of the classification this yields a transformation of the input object to a reference pose, which can be used for downstream tasks such as the computation of object support. We apply a two-step training mechanism for our framework, which alternates between optimizing over the latent transformation variables and the classifier parameters to minimize the loss function. We show that multilayer perceptrons, also known as deep networks, are well suited for this approach and achieve state of the art results on the rotated MNIST and the Google Earth dataset, and produce competitive results on MNIST and CIFAR-10 when training on smaller subsets of training data.

1. Introduction. Ulf Grenander pioneered the idea of handling the challenge presented by the geometric variability of objects in images using a generative framework based on deformable templates [GCK91, AGP91, Gre93, GM98]. The variability in a population of images is modeled via deformations applied to a prototype or template. The deformations are explicitly parameterized and represented by latent unobserved random variables. The statistical framework yields a cost function that measures the distance between the deformed template and the data. This typically has the form of a sum of squares or other likelihood based measure, usually assuming conditional independence of the pixel observations given the latent variable. There are then two interrelated challenges: given a template compute the deformation conditional on an image and estimate

Received December 18, 2017, and, in revised form, October 3, 2018.
2010 *Mathematics Subject Classification.* Primary 62H35.

Email address: jiajun@cs.uchicago.edu
Email address: amit@marx.uchicago.edu

a template from a sample of images. The first problem has been studied very extensively in a wide variety of contexts; see for example [MTY06, You10]. The problem of template estimation has received some attention; see [AT07, AAT07].

In this paper we extend these ideas from the domain of generative models to that of discriminative models. Given a classifier one can try to compute for each class and image the deformation yielding the optimal output for that class, and then label the image as the class with the highest output. In other words, instead of the distance between the template of the class and an image, our framework uses a cost function based on the class scores computed by the classifier. Furthermore, given samples of images from the different classes one can train a classifier by iterating the following two steps: first find the optimal deformation of each image to the different classes given the current parameters of the classifier and then update the parameters of the classifier given the optimal deformations of the training images to each class. Since this is most easily formulated using gradient descent-type optimization, we need classifiers that are both deformable with respect to their parameters and with respect to their input. Multilayer perceptrons also known as deep networks are then a natural choice.

Deep networks have been very successful in recent years in a wide range of classification and detection tasks. The expressiveness of these networks allows the models to explore possible variations in the data and learn visual representations that are robust to task-irrelevant variations. However, such variations need to be observed in the data when training the networks, without special design the networks do not generalize to unobserved variations in the data. The standard solution for improving the transformation invariance of the classifiers is data augmentation, where transformed versions of the original data are generated and added to the original data [FGP06, DWD15, LSBP16, DDFK16, vNP17]. This approach works well but we believe it is of interest to explore the alternative where an explicit computation of the latent deformations is performed during classification. In theory this means that the classifier does not need to be as flexible since it is not directly trying to discriminate between different instantiations of the classes, rather it only needs to learn to discriminate between images of the objects at reference instantiation. Furthermore, obtaining the reference pose of the object as part of the output of classification can assist in additional visual tasks. One such task is determining the support of the object in the image.

Spatial transformer networks (STN) [JSZ$^+$15] also transform the image as part of the classification process. They try to remove extraneous transformation variability a priori by introducing a spatial transformation module before the classification network. The image data is *first* transformed to a reference instantiation, independent of the class, and then passed on to be classified. The information the network uses to first transform the image is independent of the class and therefore is necessarily generic. We believe it is essentially computed based on the first and second order statistics of the pixel data, and as we show it is very sensitive to clutter. It is, however, of interest that the transformation network and the subsequent classification network are trained together end to end using gradient descent.

Other approaches to transformation invariance explicitly transform the network filters so that the feature maps are invariant to the selected types of transformations [WHK15,

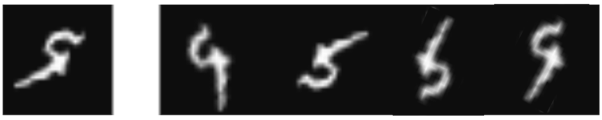


FIG. 1. A rotated image of digit five can be further rotated to look like instantiations from digit classes four, five, six, and nine.

TH16, MVKT16, ZYQJ17]. Another family of approaches tries to generalize convolutional architectures by either extending the feature space to a group space of transformations [GD14, CW16a], or warping the input so that the transformation equivariance is implicitly encoded [HV16]. These approaches are either limited to a small set of transformations, or they need to keep the models shallow because of the high computational burden required to consider additional transformations in the feature map.

In our framework, latent variables are introduced, separately for each class, to capture the transformations of the data and we apply a two-step training mechanism to alternatively optimize over the latent variables and the neural network model parameters to minimize a designed loss function. We emphasize that the latent variables are optimized for each class separately. Consequently, unlike STN that produces a single transformed version of the original input, here for each class we produce a different transformed version of a given image. The transformation is not predicted directly from the data, rather, for each class it is estimated to optimize the output of the output unit representing that class.

We show that this framework can be applied to any existing neural network architecture and offers flexibility in the types of transformation considered by the model. We apply our framework to the training of convolutional neural networks (CNN), and present competitive results on MNIST, MNIST-rot, CIFAR-10, and the Google Earth dataset. In addition to improved classification rates, we show that the estimated latent transformations indeed align the images very well, and allow us to estimate very precise object supports in the case of MNIST, and the correct object rotation in the case of Google Earth.

In Section 2 we describe related work on latent variables in the machine learning literature. In Section 3 we layout the deformable classifier algorithm. In Section 4 we describe a modification of the spatial transformer network (STN) that regresses the parameters of the transformation on the image as the STN but separately for each class. In Section 5 we describe the experiments and in Section 6 we show how these types of networks can be used to handle clutter in the case of handwritten digits.

2. Related work.

Two approaches most relevant to our proposed method are spatial transformer networks and latent SVM models.

2.1. *Spatial transformer network* (*STN*). The model of [JSZ+15] consists of a spatial transformer module that contains a localization network and a grid generator together with a classification network that can be trained end to end using stochastic gradient descent. The localization network predicts the transformation parameters based on the input image. These could be the six parameters of an affine map or a more general smooth deformation described for example through a thin plate spline. The grid generator transforms the image and the resulting transformed image is passed through the classifier. The transformation of the image is defined in a "weak" sense as follows. Consider the image domain D as a continuum where the image is defined as $\mathbf{x}(s), s \in D$. Given a parameterized family of smooth deformation functions $\phi(s, \mathbf{z})$, mapping D to D the deformed image associated with ϕ can be expressed as $\mathcal{T}_{\mathbf{z}}\mathbf{x}(s) = \mathbf{x}(\phi(s, \mathbf{z}))$. This is the original formulation in [AGP91]. Now take a smooth kernel function $K(t, u)$ defined on $D \times D$, which approximates the Dirac delta function and write

$$\mathcal{T}_{\mathbf{z}}\mathbf{x}(s) \sim \int_{u \in D} \mathbf{x}(u) K(\phi(s, \mathbf{z}), u)\, du. \tag{1}$$

The derivative with respect to $\mathbf{z}$ becomes:

$$\frac{\partial \mathcal{T}_{\mathbf{z}}\mathbf{x}(s)}{\partial \mathbf{z}} = \int_{u \in D} \mathbf{x}(u) \frac{\partial K}{\partial t}(\phi(s, \mathbf{z}), u)\, du \frac{\partial \phi(s, \mathbf{z})}{\partial \mathbf{z}}. \tag{2}$$

This formulation allows one to push the application of the deformation and the computation of derivatives onto the smooth kernel (which can be done analytically) and avoids the need to deal with explicit deformations or derivatives of the image, which is defined on a discrete pixel grid. Denote the network predicting the transformation parameters $\mathbf{z}$ as $\Psi(\mathbf{x}, \eta)$ and the subsequent classification neural network $\Phi(\cdot, \theta)$. The entire system is defined as $\Phi(\mathcal{T}_{\Psi(\mathbf{x},\eta)}\mathbf{x}(s), \theta)$. The gradients with respect to θ and η are easily propagated backwards through this network provided a module is defined to compute $\phi(s, \mathbf{z})$. Once the network is trained, i.e., $\hat{\eta}$ and $\hat{\theta}$ are estimated, an image F is passed through Ψ to obtain the predicted transformation $\mathbf{z} = \Psi(\mathbf{x}, \hat{\eta})$. Then $\phi(s, \mathbf{z})$ is computed yielding $\mathcal{T}_{\mathbf{z}}\mathbf{x}(s)$ and $\Phi(\mathcal{T}_{\mathbf{z}}\mathbf{x}(s), \hat{\theta})$ gives the classification.

In STN [JSZ$^+$15], the authors use the bilinear sampling kernel

$$K(t, u) = \max(0, 1 - |t_0 - u_0|) \max(0, 1 - |t_1 - u_1|),$$

and equation (1) is approximated as follows:

$$\begin{aligned} \mathcal{T}_{\mathbf{z}}\mathbf{x}(s) &= \sum_{u \in D} \mathbf{x}(u) \max(0, 1 - |\phi(s, \mathbf{z})_0 - u_0|) \max(0, 1 - |\phi(s, \mathbf{z})_1 - u_1|) \\ &= \sum_{|u - \phi(s,\mathbf{z})| < 1} x(u)(1 - |\phi(s, \mathbf{z})_0 - u_0|) \cdot (1 - |\phi(s, \mathbf{z})_1 - u_1|), s \in D. \end{aligned} \tag{3}$$

In recent work [SDFH18] proposes an architecture where the spatial transformer layer in the STN estimates a diffeomorphic transformation by integrating local affine transformations, showing significant improvement. Note, however, that in all these architectures the spatial transformation is computed directly from the image without knowing its class. This can create ambiguities such as in Figure 1 where we show an image of rotated digit five and how it can be rotated to look like instantiations of different digit classes. Intuitively, the spatial transformer module should recognize the class label of the image, and then extract the transformation parameters taking the class label into account. We argue that an accurate estimate of the transformation is not possible unless the image label is captured, and even given the class, one expects that some external information is needed to guide the transformation. In STN the class labels of the images are not fully captured by the spatial transformer module Ψ (otherwise we would not need a downstream network to handle the classification task) and the spatial transformer module cannot produce accurate spatial transformation for the input.

In recent work [RAS17, YKSN17] have shown that deep networks with customized architectures are able to estimate the diffeomorphism to match a pair of images. Training is performed with pairs of images where one is a training image and the second is produced by simulating a diffeomorphism of the first. In this context the transformation parameter is observed and the network can learn optimal features to predict the transformation between the two images. This is still an "easier" task than that of the STN which tries to estimate the transformation to reference pose directly from the input image—without access to a target image, or some other representation of the quality of the transformation.

2.2. *Discriminative latent variable models.* Latent variables in discriminative models have been studied in the framework of multiple instance learning (MIL), where the latent variables are used to capture the variations of instances within the same labeled bag. The MI-SVM formulation of multiple instance learning was initially proposed in [ATH03], and later reformulated as latent SVM in [FGMR10]. That work focuses on detecting objects of a given class, and a binary classifier is run across the image scoring each window x as follows:

$$f_\beta(\mathbf{x}) = \max_{\mathbf{z}} \beta \cdot \Phi(\mathbf{x}, \mathbf{z}). \tag{4}$$

Here β is a vector of model parameters, $\Phi(\mathbf{x})$ is the feature extraction function for $\mathbf{x}$, and $\mathbf{z}$ are latent values. In this model the only classifier parameters are β, and a hinge loss is used to train the model:

$$L(\beta) = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i f_\beta(\mathbf{x})), \tag{5}$$

where $y_i = 1, -1$ for object and non-object examples, respectively. Since f_β is a maximum of linear functions it is convex in β and so for negative examples for which $y_i = -1$ the summand in the cost function is convex in β. For positive examples it is not convex and the authors propose a two-step iteration. For each positive example find the optimal z_i^* given the current value of β and then optimize the convex function

$$\tilde{L}(\beta) = \frac{1}{2}\|\beta\|^2 + C\sum_{y_i=1} \max(0, 1 - \beta\cdot\Phi(\mathbf{x_i}, \mathbf{z_i^*})) + C\sum_{y_i=-1} \max(0, 1 + \max_{\mathbf{z}} \beta\cdot\Phi(\mathbf{x_i}, \mathbf{z})). \tag{6}$$

In a similar vein, in this paper we incorporate latent variables into deep neural networks to capture the transformations of the input. The input image is warped for each class separately based on the latent values that optimize its output on that class. In training, this is done for the entire batch using the old network parameter values, and then one or more gradient steps are taken to update the network parameters after the input images are warped. In our setting the classifier is not linear in its parameters and we will have no choice but to compute the optimal latent variable for each class for all examples.

3. Deformable classifiers. We consider a multiclass classifier on C classes that scores an example $\mathbf{x}$ for each class as follows:

$$f_{\beta_{\mathbf{j}}}(\mathbf{x}) = \max_{\mathbf{z}} \beta_{\mathbf{j}} \cdot \Phi_\theta(\mathcal{T}_{\mathbf{z}}(\mathbf{x})), j = 1, \ldots, C. \tag{7}$$

Here $\beta_{\mathbf{j}}$ is a vector of model parameters for class $\mathbf{j}$ and Φ_θ is a feature mapping function parametrized by θ, which in our setting is a multilayer convolutional neural network. The paramters θ are common to all classes and the final classification depends on the last layer of the network that is parameterized with the β's. The latent variable $\mathbf{z}$ is introduced to parametrize the deformations of the data and $\mathcal{T}_{\mathbf{z}}(\cdot)$ transforms the input according to the value of $\mathbf{z}$. The label $\hat{c}$ of each example is then determined by

$$\hat{c} = \arg\max_{\mathbf{j}} f_{\beta_{\mathbf{j}}}(\mathbf{x}). \tag{8}$$

For a test example, the model finds a separate optimal latent value for each class in terms of the output corresponding to that class. The class output with highest value yields the final classification. Together with that classification, we also obtain an optimal transformation of the image into reference pose.

Intuitively, in order to make the correct prediction, we want the score of the target class to be larger than the scores of the non-target classes. To achieve this, for an example $\mathbf{x}$ with label $\mathbf{y}$ we maximize the margin between $f_{\beta_{\mathbf{y}}}(\mathbf{x})$ and $f_{\beta_{\mathbf{j}}}(\mathbf{x})$ for $\mathbf{j} \neq \mathbf{y}$. Suppose we have a set of observations $\mathbf{x} = \{\mathbf{x_1}, \ldots, \mathbf{x_N}\}$ and the corresponding data labels $\mathbf{y} = \{\mathbf{y_1}, \ldots, \mathbf{y_N}\}$. We use the multiclass hinge loss as follows:

$$\mathcal{L}(\Theta) = \sum_{\mathbf{i}} \max(0, 1 + \max_{\mathbf{j} \neq \mathbf{y_i}} f_{\beta_{\mathbf{j}}}(\mathbf{x_i}) - f_{\beta_{\mathbf{y_i}}}(\mathbf{x_i})) + \lambda(\|\theta\|^2 + \sum_{\mathbf{j}} \|\beta_{\mathbf{j}}\|^2), \tag{9}$$

where $\Theta = \{\theta, \beta_1, \ldots, \beta_C\}$ are the model parameters and C is the number of classes. λ is the parameter that controls the regularization term $\|\theta\|^2 + \sum_{\mathbf{j}} \|\beta_{\mathbf{j}}\|^2$.

The key step in our method is to find the optimal instantiation of each example for each class. At first glance, it might seem that it would be simpler to forego the non-target classes and only focus on finding the optimal instantiation of the example for the target class. We note, however, that such an approach is often insufficient. Recall that an image can be transformed to look like instantiations from a non-target class, like the examples we show in Figure 1. Without competing with optimal instantiations of the data from non-target classes, the model might not be learning from the most competitive negative examples.

In order to minimize the hinge loss in equation (9), we design a two-step training mechanism. For each example, the algorithm finds the highest scoring latent values for each class based on the current model parameters. Then the algorithm optimizes over the model parameters while fixing the latent values. We outline the procedure for the two-step training algorithm in Algorithm 1 for deformable classifiers (DC).

When the latent variables form a discrete set, for example a finite set of rotations, optimization is obtained with exhaustive search (DC-ES). For continuous latent variables,

Algorithm 1 Two-Step Algorithm For Learning a Deformable Classifier (DC)

```
procedure
    Choose an initial setting for the parameters $\Theta^{\text{old}} = \{\theta^{\text{old}}, \beta_1^{\text{old}}, \dots, \beta_C^{\text{old}}\}$.
    Optimize Over z:
        $f_{\beta_{\mathbf{j}}}(\mathbf{x_i}) = \max_{\mathbf{z}} \beta_{\mathbf{j}}^{\text{old}} \cdot \Phi_{\theta^{\text{old}}}(\mathcal{T}_{\mathbf{z}}(\mathbf{x_i}))\,,\ \mathbf{j} = 1, \dots, C.$
        $\mathcal{L}(\Theta) = \sum_{\mathbf{i}} \max(0, 1 + \max_{\mathbf{j} \neq \mathbf{y_i}} f_{\beta_{\mathbf{j}}}(\mathbf{x_i}) - f_{\beta_{\mathbf{y_i}}}(\mathbf{x_i})) + \lambda(\|\theta\|^2 + \sum_{\mathbf{j}} \|\beta_{\mathbf{j}}\|^2)$
    Optimize Over Model Parameters Θ:
        $\theta, \beta_1, \dots, \beta_C = \arg\min_{\theta, \beta_1, \dots, \beta_C} \mathcal{L}(\Theta)$
    If the convergence criterion is not satisfied, then
        $\theta^{\text{old}} \leftarrow \theta^{\text{new}},\ \beta_1^{\text{old}} \leftarrow \beta_1^{\text{new}},\ \dots,\ \beta_C^{\text{old}} \leftarrow \beta_C^{\text{new}}$
        and return to line 3.
    Stop
```

we optimize $f_{\beta_{\mathbf{j}}}(\mathbf{x}, \mathbf{z})$ from equation (7) with respect to $\mathbf{z}$ by gradient descent (DC-GD). We regularize the magnitude of $\mathbf{z}$ during optimization by penalizing its distance from the identity. When the range of the continuous variable is very large, such as the 360 degree range of rotations, we initialize the gradient descent at a small set of discrete initial rotations and take the optimal value over all initializations (DC-ESGD).

In this paper, we use 2D affine transformations and thin plate spline transformations [Boo89] as the two parameterizations for ϕ. The gradient of $\phi(s, \mathbf{z})$ with respect to $\mathbf{z}$ for these two types of transformations has been implemented efficiently in the deep learning package Lasagne which we employ for our experiments.

4. Class based spatial transformer network. A different approach, which is a direct generalization of the original spatial transformer model, is to use class based spatial transformer modules (CSTN), one for each class, to directly predict values of $\mathbf{z}$ for each class during training and testing. Instead of optimizing over $\mathbf{z}$ based on the gradient of the classifier output, during training this approach optimizes over the neural network parameters that predict the class-specific transformation. Each transformation of the image is then passed through the same feature extraction network, and the classifier calculates the class scores based on the features extracted from different class-specific transformations of the image; see Figure 2. To be precise let $\Psi_c(\mathbf{x}, \theta_c)$ denote the

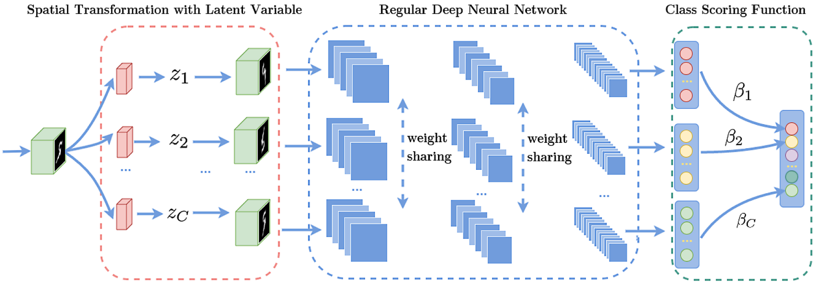


FIG. 2. Model architecture for CSTN.

network computing the transformation for each class c. Let $\Phi(\cdot,\eta)$ be the common feature extraction network; then the full CSTN computes

$$\arg\max_{c} \beta_c \cdot \Phi(\mathcal{T}_{\Psi_c(\mathbf{x},\theta_c)}\mathbf{x},\eta). \tag{10}$$

In training, as in the previous section, we alternate between updating the parameters $\eta, \beta_c, c = 1, \ldots, C$ of the classification network with the θ parameters fixed, and then keeping the classification network parameters fixed and updating the θ parameters of the transformation networks $\theta_c, c = 1, \ldots, C$. When θ is fixed the loss for example $\mathbf{x}_i$ is given by

$$L(\mathbf{x}_i, y_i, \eta, \beta) = S\left(y_i, \left[\beta_c \cdot \Phi(\mathcal{T}_{\Psi_c(\mathbf{x}_i,\theta_c)}\mathbf{x}_i,\eta)\right]_{c=1}^{C}\right),$$

where S could be the softmax loss or the hinge loss described above. When η and β are fixed, we want to optimize the spatial transformer modules $\Psi_c(\mathbf{x},\theta_c)$ separately for each class c, i.e., we maximize

$$L(\mathbf{x}_i, c, \theta_c) = \beta_c \cdot \Phi(\mathcal{T}_{\Psi_c(\mathbf{x}_i,\theta_c)}\mathbf{x}_i,\eta).$$

For a given class c we compute the optimal instantiation θ_c for each example (whether from that class or not) to make it look as close as possible to an "ideal" image of class c. As a result, the spatial transformer modules not only learn to find the optimal instantiation of each example for the target class, but also learn to compute competitive negative examples for the non-target classes.

In testing there is no need for optimization as the transformation for each class is predicted directly through Ψ_c. Again, unlike STN where only one spatial transformer module is trained for the network, this approach constructs a different spatial transformer module for each class, providing a different latent value for each class. The downstream networks from each transformer module are all tied until the final layer, where each one feeds into the corresponding class output unit. The methods DC-GD, DC-ESGD and CSTN require us to parametrize the transformation function in a form that is differentiable with respect to the latent variable so that we can use the gradient to either directly update the latent variable or update the model parameters in the spatial transformer modules.

5. Experiments.

We implement our model and perform experiments on the MNIST-rot dataset, the CIFAR-10 dataset, and the rotation angle estimation task for the Google Earth dataset.

5.1. *The MNIST-rot dataset.* The MNIST-rot dataset is a variant of the MNIST dataset [LEC+07] that consists of images from the original MNIST rotated by a random angle from 0° to 360°. The dataset contains 12000 training images and 50000 testing images.

We set the angle of rotation as the latent variable. We choose a CNN architecture which can be trained to achieve a competitive result on the MNIST dataset. The CNN architecture consists of two consecutive convolutional blocks, where each block is composed of a convolutional layer with 32 filters of size 5×5 and a maxpooling layer of size 2×2. The output of the convolutional blocks is passed to a fully connected layer with 256 units before being fed to the final layer with 10 units. We initialize the weights of the

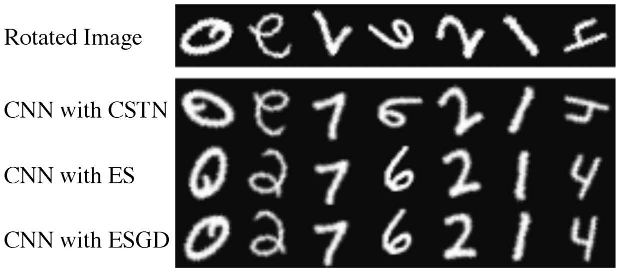


FIG. 3. Image rotations are corrected using the latent rotation angles estimated by the three optimization approaches.

CNN model by training it on a subset of the original MNIST dataset (first one hundred training images of each class). Then we train the DC model with optimal instantiations on the MNIST-rot training data. We experiment with two different approaches to optimizing over the latent variable: DC-ES and DC-ESGD and the class-based spatial transformer network CSTN. In DC-ESGD $\mathbf{z}$ is initialized at eight different rotations, and each is optimized for ten iterations using gradient descent. We choose the value of $\mathbf{z}$ that produces the highest score.

In Figure 3, we show some example images of rotated digits and their unrotated versions corrected using the latent rotation angles estimated by the three approaches. Compared to CSTN, the DC-ES, and DC-ESGD achieve better estimates of the rotated angles. The exhaustive search approach is more constrained since it can only search for a limited amount of rotations (in this case every 45 degrees). The gradient descent approach can adjust the rotation with an arbitrary angle, creating better rotation-corrected images. In Table 1, we show the error rate achieved by different models. When using CSTN to identify the latent variables, we are able to achieve an error rate of 2.64%, significantly improved from 5.71% achieved by the conventional STN. Our best result is achieved with DC-ESGD, reaching an error rate of 1.25%. The state-of-the-art result 1.2% is achieved by TI-Pooling [LSBP16], where 24 explicitly rotated versions of the images are presented to the model for training and testing.

TABLE 1. Results on the MNIST-rot dataset.

Model	Error (%)
TIRBM [SL12]	4.2
original CNN Model	4.1
STN	5.71
TI-POOLING (24 rotations) [LSBP16]	**1.2**
CSTN	2.64
DC-ES (8 rotations)	2.31
DC-ESGD (8 initial rotations)	**1.25**

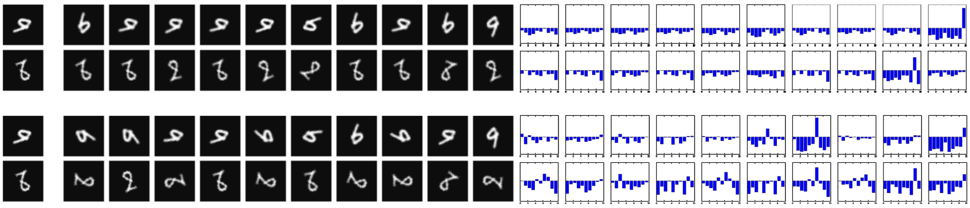


FIG. 4. Examples of rotation-corrected images for ten separate classes using a conventional CNN trained on upright digits (bottom) and a CNN trained on rotated digits using DC-ESGD (top). For each rotation-corrected image we show all ten class scores on the right, but the classifier only uses the score from the class that determined the rotation.

Our training framework allows the model to compare optimal instantiations of the image under different classes and expand the margin between the score of the target class and the highest score of non-target classes. To show why this is important, we conduct the following experiment: We first train a traditional CNN model on 60000 training images of upright digits from the original MNIST dataset with the multiclass hinge loss. The trained model can then be plugged into our framework, and, without additional training, we can use it to find the latent rotation angles of the rotated digits under each class. We compare this approach with the DC model trained with optimal instantiations. In Figure 4, we show all ten class scores for each transformed image. Note that the classifier only uses class score o_c from the image transformed based on the output for class c. We see that although the optimal latent rotation angles under the correct class labels captured by the two approaches are similar, the CNN trained using our framework effectively suppresses non-target class scores. While in the examples generated by the conventional CNN, we observe many undesired spikes of scores for non-target classes, which will lead to incorrect classifications. For DC-ESGD the example rotated 9 has a strong output for class 9 in the last display and very low outputs for any of the other classes it has been transformed to. Using the standard CNN the output for class 6 when rotated for class 6 is higher than the output for class 9 when rotated for class 9. A similar problem occurs with the rotated 8. Using the standard CNN to rotate the images achieves an error rate of 11.04%, which is far worse than any result we show in Table 1.

5.2. *MNIST.* We train our model on the original MNIST dataset [LCB98]. In order to limit the transformation invariance that can be learned from the data, we only use the first 100 images of each class from the training dataset (*called MNIST*1000). In order to capture the local deformations of the data, we use the thin plate spline transformation as the latent variables. A 4x4 grid of control points is used for the thin plate spline transformation, resulting in 32 parameters modeling the image deformations. As before, we first initialize the CNN model by training it on the MNIST1000 dataset and then train the model using our framework. In this experiment, we only use gradient descent (GD) to optimize over the latent variables.

TABLE 2. Results on MNIST-1000.

Model	Error (%)
CNN	3.17
STN	4.9
DC-GD (Thin Plate Spline)	**2.00**

In Figure 5, we show the optimal transformed image by thin plate splines for each of the ten classes. As shown in Table 2, we are able to achieve an error rate of 2.0% using our framework, which is a major improvement compared to results with the original CNN or with STN.

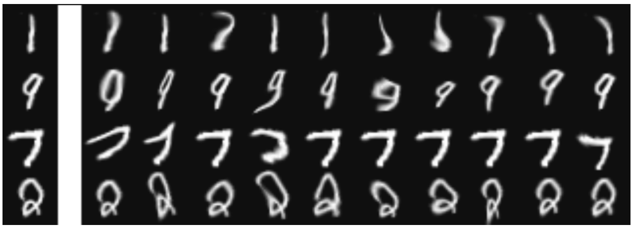


FIG. 5. Examples of optimal images deformed by the thin plate spline transformation for different classes. The original images are shown in the first column.

5.3. *CIFAR*-10. We apply our model on the CIFAR-10 dataset [KH09]. We train our model using the first 400 images of each class from the training dataset (called *CIFAR*-10(400)) and test on the original CIFAR-10 test dataset. A five-layer CNN model can achieve 28.43% test error after 4000 epochs of training on the full dataset with 5000 training images per class. We choose a CNN with the same architecture for our framework. We initialize the network by first training the CNN model on CIFAR-10(400) and then train it with the deformable classifier, again using only 400 images per class. We explore two settings: one with the angle of rotation as the latent variable for the model and the other with translation and scale as the latent variables. In Figure 6, we show the optimal transformation via translation and scaling for objects of CIFAR-10 images. We show a 2.9% increase of model performance using a CNN with latent translation and scaling, which is even comparable with some of the semi-supervised approaches reported in the literature that use a large complementary unlabeled training set.

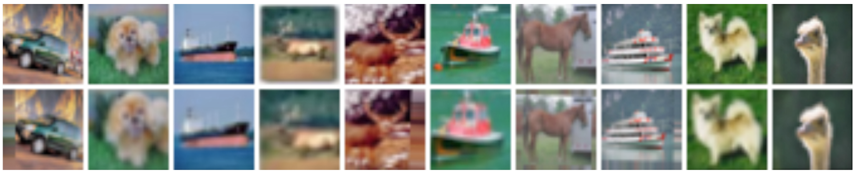
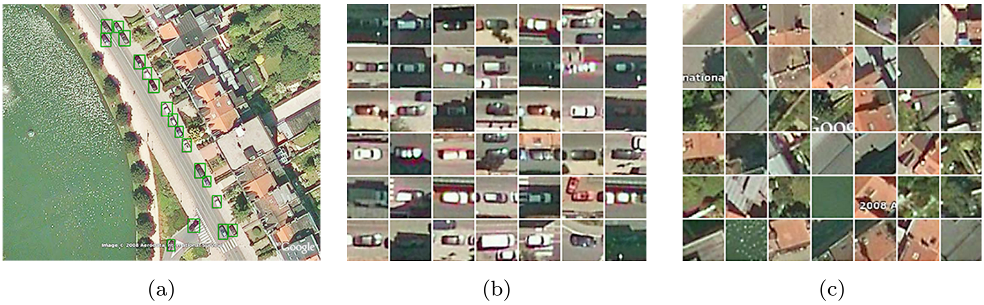


FIG. 6. *Top:* Example images from the CIFAR-10 dataset; *Bottom:* Images translated and scaled by our model.

TABLE 3. Experiment Result on CIFAR-10(400)

Model Description	Architecture	Error (%)
DCGAN (semi-supervised approach) [RMC15]		26.2(±0.4)
Exemplar-CNN (semi-supervised approach) [DSRB14]	64c5-128c5-256c5-512f	24.6(±0.2)
Exemplar-CNN (semi-supervised approach) [DSRB14]	92c5-256c5-512c5-1024f	23.4(±0.2)
Steerable-CNN [CW16b]	14 layers, 4.4M params	24.56
CNN Baseline	64c3-64c3-128c3-128c3-256c3-256f, 1.6M params	28.43
CNN with DC-ESGD (Rotation)	64c3-64c3-128c3-128c3-256c3-256f, 1.6M params	27.9
CNN with DC-GD (Translation, Scale)	64c3-64c3-128c3-128c3-256c3-256f, 1.6M params	25.53

5.4. *Google Earth dataset.* We train our model on the Google Earth dataset [HK08], which contains aerial photos of streets with bounding boxes around the vehicles. Henriques et al. [HMCB14] also add angle annotation for each vehicle as a supplement to the dataset. The dataset contains 697 vehicles in 15 large images, where the first ten



(a) (b) (c)

FIG. 7. (a): An example of training images from the Google Earth dataset. (b) and (c) are examples of car images (car front point to the right) and background images we use for training a detection model for horizontal cars.

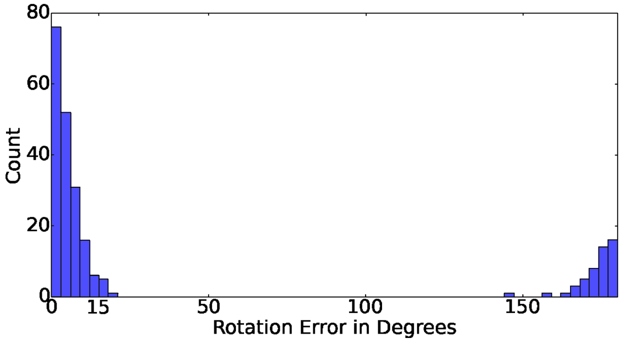


Fig. 8. Histogram of rotation errors when estimating the rotation angles between −180° to 180°.

images are used for training and the rest for testing. The task here is to estimate the rotation parameter for each vehicle in the image. We first learn a horizontal car detection model by training a classical CNN model to discriminate between horizontal car images and background images. In Figure 7, we show some image examples for training this base detection model. We use this model to initialize the DC-ESGD training algorithm, which is applied to images of rotated vehicles cropped from the training images. Note that we do not make use of the provided angle annotation as input to this algorithm. We then use the trained latent variable model to estimate the rotation angles of the vehicles by finding the latent rotation parameters that give the maximal values for the score function.

We also build a baseline 3-layer CNN model following the description in [HV16], where the last layer of the network contains one node to regress the target rotation angles of the vehicles (in radians). The results are shown in Table 4. We find that the CNN model with optimal instantiations outperforms the baseline model by a large margin, and most of the rotation errors are contributed by the cases where the car fronts are mistaken for the car rears. More specifically, as we show in Figure 8, 77% of the data are predicted with less than 15° of rotation error while 22% are predicted with more than 150° of rotation error. If we ignore the difference between the front and the rear of the car and relax our problem by estimating the rotation angles between −90° to 90°, we achieve an average test rotation error of 4.87°.

Note that the CNN for regressing the rotation angle result from [HV16] shown in Table 4 uses a different approach to calculate the rotation errors. Let us denote by $\alpha_i, \hat{\alpha_i} \in (-\pi, \pi)$ the ground truth and the predicted value of the angle, respectively, for example i. Henriques et al. [HV16] define the rotation error as $e_i = \frac{\pi}{2} - \Big| |\alpha_i \bmod \frac{\pi}{2} - \hat{\alpha_i} \bmod \frac{\pi}{2}| - \frac{\pi}{2} \Big|$. We believe a better metric would be $e_i = \pi - \Big| |\alpha_i - \hat{\alpha_i}| \bmod 2\pi - \pi \Big|$ if $\alpha_i, \hat{\alpha_i} \in (-\pi, \pi)$, and $e_i = \frac{\pi}{2} - \Big| |\alpha_i - \hat{\alpha_i}| \bmod \pi - \frac{\pi}{2} \Big|$ if $\alpha_i, \hat{\alpha_i} \in (-\pi/2, \pi/2)$. We provide the error result of the CNN for regression based on our calculation.

Table 4. The average rotation errors of different models.

Model Description	Average rotation error (degree)
CNN for regression [HV16]	28.87
Warped-CNN [HV16]	26.44
CNN for regression ($-180°$ to $180°$)	63.7
CNN for regression ($-90°$ to $90°$)	43.1
CNN with DC-ESGD ($-180°$ to $180°$)	37.8
CNN with DC-ESGD ($-90°$ to $90°$)	4.87

6. Robustness to clutter. Handling clutter in an image is of paramount importance, as clutter can lead to significant degradation of classifier performance if not observed during training. We thus investigate the sensitivity of the DC approach to different types of clutter, which are not observed in the training data. We employ two types of clutter models.

Flanking digits: We put two digits on the two sides of the original digit and crop the image, so we have parts of the flanking digits as clutter. This kind of clutter is very common when dealing with digit sequence recognition.

Random clutter: We randomly select small image patches from digit images and place them randomly around the original digits. These patches contain digit parts such as strokes and curvatures.

In Figure 9 we show some examples of images with the two different clutter types. Note that nearby clutter will not touch or overlap with the original digit in the center. In the first two rows of Figure 10 are the reference poses recovered by our approach for images with flanking digit clutter. Only the reference poses of the images under the correct class labels are shown here. The model is able to adjust the center digits to obtain the preferred poses. It is worth noting that the digits in our training data have the same size as the digits in the test data.

In the first two rows of Figure 11 we show the reference poses recovered by our approach for images with random clutter surrounding the target objects. Similarly, the model can adjust the pose of the target object in the center regardless of the surrounding random clutter. It is worth noting that the reference poses estimated for the objects with surrounding random clutter are different from those captured for the objects with flanking digits.

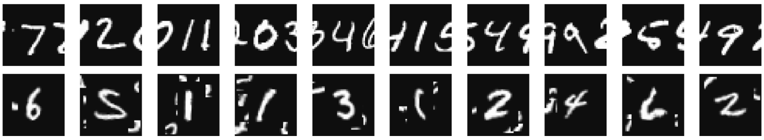


Fig. 9. Sample images of two different types of clutters: flanking digits (top) and random clutter (bottom).

6.1. *Declutter images with object class support maps.* After the reference poses are estimated, the surrounding clutter still exists and can affect the outcome of the classifier. The DC framework is then used to estimate support masks for the objects, which are used to eliminate the clutter.

In Figure 12, we show the comparison between the mean images of the original handwritten digits in the training set and the pose-adjusted handwritten digits recovered by the thin plate splines (TPS). As nuisance transformations in the data are removed to obtain the reference pose of the object, it is clear that the mean images of the pose-adjusted digits are much sharper than the mean images of the original digits and can be used to determine an object support map.

In the presence of clutter, if the object label of the image is known, we can apply the support maps of the correct class to the pose-aligned image with clutter and obtain the decluttered image, as shown in Figures 10 and 11. Note that this decluttering step is naturally achieved with the pre-trained model. This is very useful when dealing with tasks where the objects in the training images have clean background while the objects in the testing images are surrounded by clutter.

Note that in Figure 11, we observe that some parts of the objects are cut out by the support masks (for example, digit 9 in the seventh column and digit 7 in the eighth column). The shapes of the recovered reference poses cannot completely match the support masks of the corresponding class, indicating that the surrounding clutter is affecting the classification result.

6.2. *Classifying with clutter using object support maps.* The results shown in Figures 10 and 11 assume knowledge of the correct class. This is not known in the actual classification setting. Since in our approach we estimate the reference pose for each class separately we can apply the support map of that class before passing it to the downstream network to get the output score for that class. The class output with highest value yields the final classification. When trained on the MNIST1000 dataset with a clear background and tested on the test dataset with flanking digit clutter, our approach improves the classification accuracy rate from 89.82% to 91.07% when we remove the

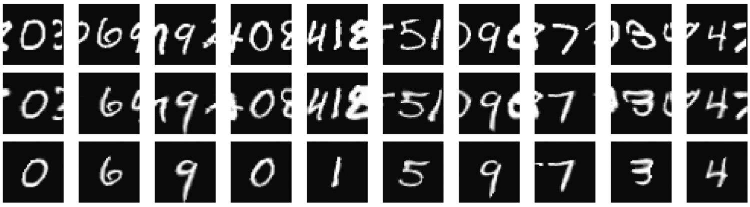


FIG. 10. Examples of the original images with flanking digit clutter are shown in the first row. The corresponding recovered reference poses under the correct class labels are shown in the second row. The decluttered images extracted by applying object class support are shown in the third row using a decluttering approach described in Section 6.1.

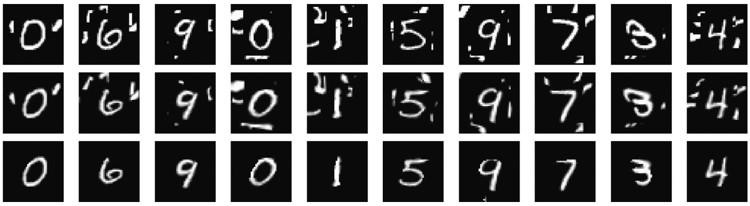


FIG. 11. Examples of the original images with random clutter are shown in the first row. The corresponding recovered reference poses under the correct class labels are shown in the second row. The decluttered images extracted by applying object class support of the correct class are shown in the third row using a decluttering approach described in Section 6.1.

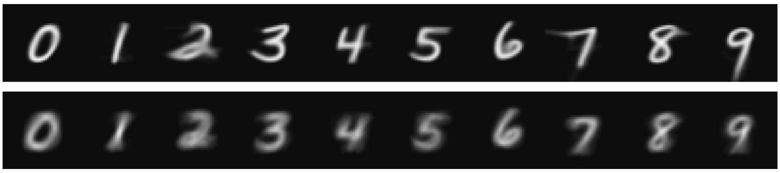


FIG. 12. Mean images of handwritten digits (bottom) and pose-adjusted handwritten digits (top).

clutter from test images using the support maps. However, when performing the same experiment with random surrounding clutter, the classification accuracy rate drops from 88.59% to 86.91%. Our approach is less robust to random surrounding clutter.

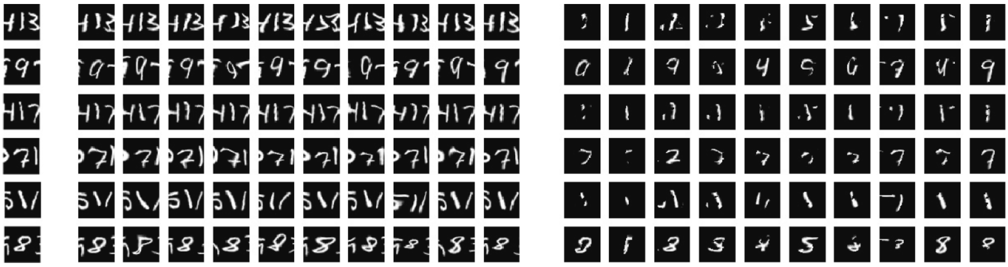


FIG. 13. We show examples of misclassified digits and the corresponding optimal images captured for the corresponding images for different classes (in the middle). On the right, we show the corresponding decluttered images that we feed to the downstream network to produce class scores for classification.

The mistakes made by the decluttering approach are shown in Figure 13. There are two types of issues:

- The subset problem: one class, say c, after transformation, can look like a subset of another class d. On an image of class d, once the support map for c is applied only the subset is visible and the image gets a high score for class c. For instance, in the second row of Figure 13, a digit 9 can look like digit 0, digit 4 and digit 7 after we deform it and apply the support maps. Note that our classification model will produce a class score for each class separately and label the test example with the class that has the highest score. Therefore, having decluttered images look like images from a different class other than the target class during the classification stage would confuse the classifier.
- Some mistakes are caused by some undesired deformations. For instance, in the examples we show in the first row of Figure 13, we observe in the sixth column that clutter from the nearby region gets pulled to the digit in the center to form a new object that looks like a digit 5. After we apply the support map to the image and remove the clutter, this image looks exactly like a digit 5. This is caused by too much flexibility of the deformation allowed in the thin-plate spline, which we can alleviate by regularizing on the degree of deformation.

As explained above, for each image, we directly feed the decluttered images for different classes to the downstream classifier for classification. The class scores are produced for each decluttered image separately, and the class that produces the highest score will be picked to determine the label of the example. Since the classification model only observes the decluttered image for a certain class, without being aware of the decluttered images for other classes or what got masked out in the original image, the model simply does not have the information on whether a certain object class can best explain the scene in the original image.

To resolve this, we apply a two-step mechanism for training and testing the images, *with no clutter observed in training.* We train a regular deformable classifier f and estimate a support map for each class. Then for each training image the optimal instantiation for each class is computed using f and the corresponding support map applied yielding ten transformed and cropped images, as show in Figure 14. For each training image we stack these ten transformed and cropped images and train a regular CNN to classify the label of the example based on the stack of input images. This time, since the model is able to observe the optimal deformed and decluttered images from all the classes, it has richer information on what gets masked out by the support maps, and it can better resolve the subset problem. By applying this two-step mechanism to classifying images with clutter, we achieve a classification accuracy of 93.47% on the test images with flanking digit clutter and a classification accuracy of 91.86% on the test images with random surrounding clutter. These are, respectively, 2.3% and 4.95% higher than the approach without the two-step mechanism.

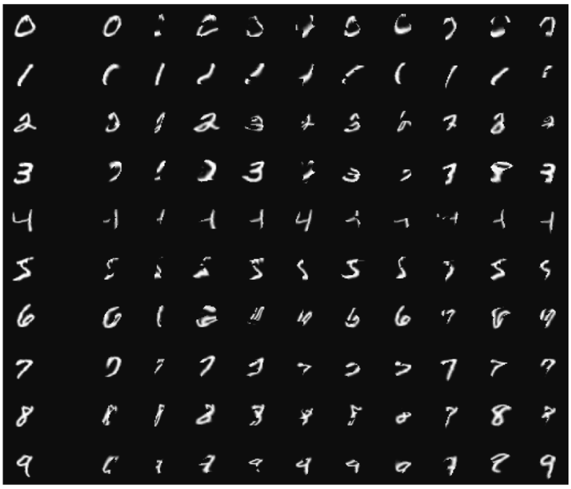


FIG. 14. The stack of ten images produced from each training image, after the optimal instantiation is computed for each class and the corresponding class support mask is applied. The leftmost column is the original training image.

7. Discussion. In this work, we propose a framework for training deep neural networks with optimal instantiations of the data. By introducing latent variables to parameterize the transformation of the data for each class, our approach is able to obtain the reference pose for the object that is being classified, and consequently can achieve better classification rates with smaller training sets. We show that such an approach can be applied to any existing neural network architecture and is compatible with general types of transformations including rotation, translation, scaling, and local deformations. This presents a non-generative approach to estimating latent transformations, but can be used to estimate templates for the different classes by averaging over the pose-corrected images. When generative methods are used for classification the templates are needed in order to compute the likelihood of each class for a given test image. Here the templates are not needed for classification but can be used to estimate the object support and to identify object parts. Furthermore, by introducing discrete latent variables we believe it should be possible to estimate clusters or mixture components for the different object classes, thus refining the estimated templates. One clear advantage of generative modeling is that for each class only examples of that class are needed to estimate the template and the distribution over the latent variables. The disadvantage of such modeling is the inadequacy of the noise models, which typically need to assume conditional independence in order for the model to be computationally tractable. In our setting all class labels need to be known in order to update the parameters of the network. This is essentially determined by the particular multiclass hinge loss we use. We note that it is also possible to use 'one-against-the-rest' hinge losses, where for each class we are estimating a two class classifier. In that case we would be learning the deformable classifier for each class separately, and implicitly a template for that class and even a distribution over deformations, avoiding the need to provide a generative model for the images. To summarize, much of the important information about the distribution of samples in each class that is obtained from generative modeling can be obtained in the framework proposed here, provided there is a classification cost and data available to evaluate this cost.

References

[AAT07] S. Allassonnière, Y. Amit, and A. Trouvé, *Towards a coherent statistical framework for dense deformable template estimation*, J. R. Stat. Soc. Ser. B Stat. Methodol. **69** (2007), no. 1, 3–29, DOI 10.1111/j.1467-9868.2007.00574.x. MR2301497

[AGP91] Y. Amit, U. Grenander, and M. Piccioni, *Structural image restoration through deformable template*, Journal of the American Statistical Association **86** (1991), no. 414, 376–387.

[AT07] Y. Amit and A. Trouvé, *Pop: Patchwork of parts models for object recognition*, Intl. Jour. of Comp. Vis. **75** (2007), 267–282.

[ATH03] S. Andrews, I. Tsochantaridis, and T. Hofmann, *Support vector machines for multiple-instance learning*, Advances in neural information processing systems, pages 577–584, 2003.

[Boo89] F. L. Bookstein, *Principal warps: Thin-plate splines and the decomposition of deformations*, IEEE Transactions on pattern analysis and machine intelligence, 11(6):567–585, 1989.

[CW16a] T. S. Cohen and M. Welling, *Group equivariant convolutional networks*, arXiv preprint arXiv:1602.07576 (2016).

[CW16b] T. S. Cohen and M. Welling. *Steerable cnns*, arXiv preprint arXiv:1612.08498 (2016).

[DDFK16] S. Dieleman, J. De Fauw, and . Kavukcuoglu, *Exploiting cyclic symmetry in convolutional neural networks*, arXiv preprint arXiv:1602.02660 (2016).

[DSRB14] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, *Discriminative unsupervised feature learning with convolutional neural networks*, Advances in Neural Information Processing Systems, pages 766–774, 2014.

[DWD15] S. Dieleman, K. W. Willett, and J. Dambre, *Rotation-invariant convolutional neural networks for galaxy morphology prediction*, Monthly notices of the royal astronomical society, 450(2):1441–1459, 2015.

[FGMR10] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, *Object detection with discriminatively trained part-based models*, IEEE transactions on pattern analysis and machine intelligence, 32(9):1627–1645, 2010.

[FGP06] B. Fasel and D. Gatica-Perez, *Rotation-invariant neoperceptron*, Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, volume 3, pages 336–339. IEEE, 2006.

[GD14] R. Gens and P. M. Domingos, *Deep symmetry networks*, Advances in neural information processing systems, pages 2537–2545, 2014.

[Gre93] U. Grenander, *General pattern theory*, Oxford Mathematical Monographs, The Clarendon Press, Oxford University Press, New York, 1993. A mathematical study of regular structures; Oxford Science Publications. MR1270904

[GCK91] U. Grenander, Y. Chow, and D. M. Keenan, *A pattern theoretical study of biological shape*, Springer Verlag, New York, 1991.

[GM98] U. Grenander and M. I. Miller, *Computational anatomy: an emerging discipline*, Quart. Appl. Math. **56** (1998), no. 4, 617–694, DOI 10.1090/qam/1668732. Current and future challenges in the applications of mathematics (Providence, RI, 1997). MR1668732

[HK08] G. Heitz and D. Koller, *Learning spatial context: Using stuff to find things*, European conference on computer vision, pages 30–43. Springer, 2008.

[HMCB14] J. F. Henriques, P. Martins, R. F. Caseiro, and J. Batista, *Fast training of pose detectors in the Fourier domain*, Advances in neural information processing systems, pages 3050–3058, 2014.

[HV16] J. F. Henriques and A. Vedaldi, *Warped convolutions: Efficient invariance to spatial transformations*, arXiv preprint arXiv:1609.04382 (2016).

[JSZ$^+$15] M. Jaderberg, K. Simonyan, A. Zisserman, et al., *Spatial transformer networks*, Advances in Neural Information Processing Systems, pages 2017–2025, 2015.

[KH09] A. Krizhevsky and G. Hinton, *Learning multiple layers of features from tiny images*, 2009.

[LCB98] Y. LeCun, C. Cortes, and C. J. C. Burges, *The mnist database of handwritten digits*, 1998.

[LSBP16] D. Laptev, N. Savinov, J. M. Buhmann, and M. Pollefeys, *Ti-pooling: Transformation-invariant pooling for feature learning in convolutional neural networks*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 289–297, 2016.

[LEC$^+$07] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, *An empirical evaluation of deep architectures on problems with many factors of variation*, Proceedings of the 24th international conference on Machine learning, pages 473–480. ACM, 2007.

[MVKT16] D. Marcos, M. Volpi, N. Komodakis, and D. Tuia, *Rotation equivariant vector field networks*, arXiv preprint arXiv:1612.09346 (2016).

[MTY06] M. I. Miller, A. Trouvé, and L. Younes, *Geodesic shooting for computational anatomy*, J. Math. Imaging Vision **24** (2006), no. 2, 209–228, DOI 10.1007/s10851-005-3624-0. MR2227097

[RMC15] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, arXiv preprint arXiv:1511.06434 (2015).

[RAS17] I. Rocco, R. Arandjelovic, and J. Sivic, *Convolutional neural network architecture for geometric matching*, Proc. CVPR, volume 2, 2017.

[SDFH18] N. S. Detlefsen, O. Freifeld, and S. Hauberg, *Deep diffeomorphic transformer networks*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4403–4412, 2018.

[SL12] K. Sohn and H. Lee, *Learning invariant representations with local transformations*, arXiv preprint arXiv:1206.6418 (2012).

[TH16] D. Teney and M. Hebert, *Learning to extract motion from videos in convolutional neural networks*, arXiv preprint arXiv:1601.07532 (2016).

[vNP17] N. van Noord and E. Postma, *Learning scale-variant and scale-invariant features for deep image classification*, Pattern Recognition, 61:583–592, 2017.

[WHK15] F. Wu, P. Hu, and D. Kong, *Flip-rotate-pooling convolution and split dropout on convolution neural networks for image classification*, arXiv preprint arXiv:1507.08754 (2015).

[YKSN17] X. Yang, R. Kwitt, M. Styner, and M. Niethammer, *Quicksilver: Fast predictive image registration–a deep learning approach*, NeuroImage, 158:378–396, 2017.

[You10] L. Younes, *Shapes and diffeomorphisms*, Applied Mathematical Sciences, vol. 171, Springer-Verlag, Berlin, 2010. MR2656312

[ZYQJ17] Y. Zhou, Q. Ye, Q. Qiu, and J. Jiao, *Oriented response networks*, arXiv preprint arXiv:1701.01833 (2017).