THE UNIVERSITY OF CHICAGO

TRANSFORMATION INVARIANCE AND EQUIVARIANCE IN DEEP LEARNING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
JIAJUN SHEN

CHICAGO, ILLINOIS
DECEMBER 2017

In dedication to my parents Jian Shen, Lan Shi and my wife Qile He for their love, endless support, encouragement, and sacrifices.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my principal advisor, Professor Yali Amit for his advice, ideas, and patience when guiding me through my graduate studies. His views on computer vision and deep neural networks, as well as his emphasis on clear statistical modeling, have been very influential to my research. From Yali, I learned not only the knowledge of statistical modeling and computer vision but also the scientific ways of thinking and tackling problems.

I thank Professor Greg Shakhnarovich and Professor Risi Kondor for being on my committee and for providing me valuable suggestions on my work. My appreciation also extends to the members of the Amitgroup - Lian Huan Ng, Marc Goessling, Gustav Larsson, Mark Stoehr and Kendra Burbank for all the inspiring discussions and collaborations. I thank the faculty and staff members of the computer science department and the statistics department for all the help, guidance and support over the last five years.

Finally, I would like to give my special thanks to my friends and family. Their unconditional love and support have given me the strength and determination to overcome the difficulties along the way.

ABSTRACT

A major challenge for object recognition is to correctly perceive the image objects despite the extraneous variations in the data such as shifting, rotation, deformation, etc. It would be much easier for the vision tasks if such task-irrelevant transformation variabilities were removed from the data. The recent success of deep learning approaches has its roots in the ability to build feature representations that are invariant to variations caused by nuisance factors. The expressiveness of deep networks allows the models to disentangle the underlying factors of variations in the data and the training signals guide the models to learn feature representations that are robust to task-irrelevant variations. However, such variations need to be observed from the training data. Otherwise, a traditional deep neural network without special architectural design would not generalize to these variations. To address this concern, we study the problem of achieving transformation invariance and equivariance in deep learning.

We show how some existing approaches, such as the stacked statistical model with rotatable features and the spatial transformer network, are imperfect at learning feature representations that are invariant or equivariant to transformations. To search for an alternative, we develop a training mechanism to learn transformation-invariant feature representations, where feature maps of canonical images are used as soft targets to guide a deep neural network to produce the same feature representations even when the input images are transformed. As a result, our framework can obtain transformation-invariant feature representations and makes it possible to take advantage of unlabeled data that contains an enormous amount of variations. Additionally, we seek architectural changes to the existing deep learning models and propose a framework for training deep neural networks with optimal instantiations. By introducing latent variables to parametrize the transformations of the data for each class, our approach is able to obtain the optimal instantiations while training for the downstream tasks. Another direction we explore is the use of 3D CAD models to render 2D images as a data augmentation approach. Rendering 2D images from 3D models allows for a more compact way of representing an object class and the models are able to observe more data variations during training. Competitive experimental results are demonstrated by these methods, and our analysis shows that they can be promising directions to achieve transformation invariance and equivariance in deep learning.

CHAPTER 1
INTRODUCTION

Learning transformation-invariant representations in deep learning is particularly important, as it helps the model to perceive image objects to be the same despite the task-irrelevant transformations. Good generalization performance can be achieved even when the model is applied to transformed data if the learned feature representations are invariant to extraneous data transformations. On the other hand, the feature representations in the intermediate layers should only be invariant to transformations within a desired range as opposed to fully invariant, as the poses and the structures of the local features need to be preserved for layers at a later stage [15, 16]. Therefore, many have sought to learn the equivariance of feature representations, i.e., instead of learning transformation-invariant representations, one can learn representations that change in a predictable way under the transformations of the input [48]. As a result, operations on a feature map in the representation space, such as the max-pooling operation, can be designed to achieve transformation invariance within a desired range.

The main theme of this dissertation is to discuss approaches for learning transformation invariance and equivariance for visual representations through deep learning. In this chapter, we first discuss the motivation of learning transformation invariance and equivariance in deep learning. We summarize some prior work on this topic in Section 1.2, and an overview of the dissertation chapters is provided in Section 1.3.

1.1 Motivation

Let us consider a regular feedforward deep neural network that is proposed to solve an object classification task. Such a deep neural network usually produces a sequence of representations by feeding the input through multiple layers, and the final feature representations are passed to the last layer for classification.

More specifically, an input image $x \in \mathbb{R}^{H \times W \times C}$ is mapped to a feature map in the representation space by network Φ as $\Phi(x) \in \mathbb{R}^d$. Let $T_g x$ denote the result of applying the transformation g on the input image x. We want to study how the representations change upon transformations of the input image, i.e., what is the relationship between $\Phi(x)$ and $\Phi(T_g x)$. In particular, we call Φ equivariant to the transformation g if we have the following:

$$\Phi(T_g x) = T'_g \Phi(x) \tag{1.1}$$

where $T'_g : \mathbb{R}^d \to \mathbb{R}^d$ is a map that captures how the feature maps will change after the input image is transformed. That is, if we transform the input by the transformation g and then pass it to the network, the output should be the same with passing the input to the network and then map the feature representation by T'_g. In particular, if we have T'_g as identity, then Φ is transformation invariant under the transformation g.

It is desirable to achieve feature representations that are invariant to task-irrelevant transformations so that the final feature representations will contain nothing but the task-related variations. As a result, the classifier is not required to memorize a large number of feature representations of transformed copies of the same input, making it easier to train. However, in some cases, feature representations should only be invariant to transformations within a certain range and still retain discriminative properties when the transformations are beyond a specific range. Therefore, it is also crucial for the deep learning approaches to obtain transformation equivariance. When the feature representations are equivariant to a certain transformation, the range of the resulting feature representations for the input after the transformation is tractable.

Some existing deep learning architectures already have such properties for certain transformations. For example, by adopting weight sharing in the network architecture, the convolutional neural network effectively captures the translation equivariance, as all the convolution layers in the network are translation equivariant. Shifting the input and then passing it through the convolution layer produces the same feature map as passing the input through the convolution layer followed by a shifting operation. This also leads to a more efficient parametrization and allows a pooling operation to operate on the feature map to achieve local translation invariance. However, the convolutional neural network only captures the translation equivariance by design. Invariance and equivariance to other types of transformations including rotation, deformation, etc. cannot be naturally obtained unless enough data is observed when training the model. To address these problems, many approaches have been proposed, and we will discuss some of them in the next section.

1.2 Prior Work

In this section, we discuss some of the previous work on learning transformation invariance and equivariance with deep learning architectures. There are three major families of approaches in the literature on this topic, including the data augmentation approach and approaches that require architectural changes to the existing deep learning models to either transform the input data or transform the filters in the models. We list some of these approaches in the following.

1.2.1 Data Augmentation Approach

Data augmentation approaches are the most common and convenient way to handle the transformation invariance and equivariance. Such approaches usually require us to either create or collect different transformed data to supplement the original dataset. As a result, the model is able to observe more data variations that do not exist in the original dataset, and it is therefore more robust to the nuisance factors of variations in the data [71, 43].

For example, the rotation-invariant neoperceptron model [20] creates multiple rotated versions of images and feeds them to a CNN with filters shared across different orientations so that feature maps for separate rotated versions of the image are obtained. A blurring layer is then designed to reduce the number of the feature maps by pooling together within a certain range of orientations. The representations are gradually pooled together using blurring layers, yielding fully invariant representations at the output. In a similar vein, the TI pooling network [45] passes multiple transformed versions of the input separately through the network to get feature representations for each transformed instance. The feature representations from transformed versions of the same input are aggregated together by a max pooling operation to achieve transformation invariance. The aggregated feature representations are later passed to the rest of the network for the downstream task. A similar approach is designed by Dieleman et al. [18], where multiple rotated and flipped versions of images are created and passed to the weight-shared convolutional layers like the TI-pooling network. Instead of being aggregated by a max pooling operation in the TI-pooling network, the resulting representations are concatenated and sent to a stack of dense layers for downstream applications. It is later extended in [17] by exploiting the cyclic symmetry of the feature maps in CNN, where four different cyclic operations including slicing, pooling, rolling, and stacking, are introduced to handle the cyclic symmetry in the representation space. Among these cyclic operations, the slicing operator first augments the original training data by stacking together four evenly spaced rotated copies of the input example, and the pooling/stacking operator combines the output from different rotated versions of the example using a permutation-invariant pooling function or a concatenating operation. The cyclic rolling operator designs a way to organize and re-align the feature maps from different rotated copies of an image, and then it stacks the feature maps together along the feature dimension. As a result, the stacked feature representations of the images become equivariant to rotations of 90°, 180° or 270°. The aggregated transformation-equivariant feature representations can be further passed to the downstream layers to handle a particular task.

A few approaches propose some changes to the deep learning framework that are combined with the data augmentation approach. The Rifd-CNN [14] proposes a rotation-invariant layer that is trained by imposing a regularization constraint on the objective function that enforces the invariance of the feature maps before and after rotating the image. Noord and Postma [77] propose an ensemble of convolutional neural networks, where each convolutional network is associated with a different scaled version of the input. The output feature maps of separate CNNs will be aggregated for further prediction. As a result, such aggregated feature maps become scale-invariant.

Although the data augmentation approaches naturally enjoy the advantage of observing more variations in data, the drawbacks of these approaches are also obvious: In general, the data augmentation methods take longer to train, as more transformed versions of the training data need to be created and observed during training. Moreover, these methods are usually constrained as they can only consider a fixed set of global transformations of the input images.

1.2.2 Transforming the Data

Another common approach is to design an operation to transform the input data or the feature maps before feeding them to the rest of the deep architecture. Special transformation operations can be designed to either remove the task-irrelevant variations by recovering the reference forms of the data or warp the input data so that a regular deep neural network can achieve transformation invariance or equivariance to a certain transformation.

For example, as we will discuss in Section 3.2, the spatial transformer network [38] can transform each input image based on a transformation matrix estimated by a spatial transformer module within the network. As a result, a reference pose of an image can be recovered before the model passes it to the rest of the network for classification. The warped convolution network [32] shows that a specially designed generalized convolution operator could be equivariant to a prefixed transformation. By applying transformations repeatedly to a pivot point to obtain a 2D warp grid for one or two commutable transformations, and a warped image is created based on the 2D warp grid. The warped input can be further passed to a regular convolutional neural network to handle the downstream task.

These approaches have the advantage of being more efficient in training, as the models do not need to be trained on multiple transformed versions of the input images. By recovering the reference poses of the input images, the downstream network is no longer required to be invariant to transformations. However, as we will find out in Section 5.3, it is difficult to estimate the desired reference pose of an input image, especially when the label of the image is unknown. The spatial transformer network tries to warp the image based on the transformation parameters regressed by the spatial transformer module, but we show that such transformations are not accurate unless the class labels are already known. The warped convolutional network warps the input images so that the transformation equivariance are implicitly encoded in the input. However, the model limits the number of transformations it can consider at the same time, which is suboptimal.

1.2.3 Transforming the Filters

In addition, special designs of filters in deep learning architectures can also make it possible to produce feature representations that are invariant or equivariant to transformations.

Kavukcuoglu et al. [40] propose a model that tries to learn transformation-invariant filters using sparse coding. It builds a 2D topographic map for the filter coefficients and pre-wires overlapping subwindows of filter coefficients to be pooled together. During training, a sparsity penalty on the sum of each neighborhood's activations is designed to encourage the activations across subwindows to be sparse and the coefficients within each subwindow to be similar. As a result, filters that are pooled together end up extracting similar features. The responses of nearby neighbors in the topographic map are pooled together to achieve transformation invariance. This idea is further extended for convolutional neural networks in the tiled convolutional neural network [55]. Instead of sharing weights across the entire image space, tiled convolutional neural network leaves nearby convolutional filter weights untied and only tying the weights of the units that are k steps away. Activations of units that have untied weights are pooled together in the next layer, allowing the network to learn a more complex range of invariances. A similar approach is investigated by Teney and Hebert [75], where filters of each convolutional layer are separated into groups. Filters within each group are constrained to be rotated versions of each other, and only the weights of the filters with canonical orientation will be learned. Oriented response network [82] incorporates the active rotating filters (ARFs) into deep convolutional neural networks. An active rotating filter is defined on a spatial grid with R rotated channels, and a rotated version of the filter can be obtained by a step of coordinate rotation using bilinear interpolation and a step of orientation spin by adjusting the rotation channel. When convolved with the input data, an ARF will create several rotated versions of the base filter and produce a feature map with multiple orientation channels. The rotation equivariant vector field network [50] takes a similar approach. It applies different rotated versions of the filters to the input and returns a vector field that represents the magnitude and angle of the highest scoring orientation at the spatial location. Instead of using a polar representation for the feature vectors, one can represent the feature vector in the Cartesian coordinate system, and the convolution of the vector fields can be computed separately in each component. Although these approaches demonstrate a promising direction to achieve transformation invariance by transforming or organizing the filters with special designs, the drawbacks of these approaches are also obvious: Since filters usually have smaller sizes, it is not clear how to apply complex transformations to the filters. As a result, invariance to complex transformations cannot be easily obtained by this approach. For filters that capture both the magnitudes and the directions of the activations, they usually have to make a compromise on the model complexity by either keeping the model shallow or limit the number of transformations.

Filters can also be designed to construct a symmetry space for the representation space. The deep symmetry network [22] achieves this by proposing a generalization of CNN that can form feature maps over a symmetry space in addition to the grid space as in the traditional CNN. Because extending the convolution operation to the symmetry space leads to a high-dimensional feature map, the authors propose to sample the symmetry space by evaluating at N control points that are local maxima of the feature representation in symmetry space found by the Gauss-Newton optimization. A pooled feature map is computed according to the N local optima using kernel-based interpolation method. The group equivariant convolutional network [15] follows the same idea of building convolutional neural networks on the symmetry space by constraining to a smaller discrete symmetry group (with flipping and four 90° rotations). As a result, the computation of the feature representations is more tractable. A subgroup pooling operation is also introduced to allow for invariance to transformations within a pooling range. To pool the full feature space G, it selects a subgroup H as the pooling region, which leads to a feature map on the quotient space G/H. These papers provide a foundation for approaches that construct symmetry spaces for the representation space. However, these approaches are still constrained because of the limited transformations they can consider.

1.3 Overview

In this dissertation, we want to explore some approaches that have been used in the literature to learn visual representations that are invariant or equivariant to transformations, and we propose several new approaches that can achieve such properties.

Some baseline models are investigated in Chapter 2. We first present the stacked statistical model with rotatable features proposed in [54], where a 3-layer layer-wise trained mixture model is introduced with the option of constructing an organized feature space that is equivariant to cyclic transformations. The simplicity and interpretability of the statistical model are appealing, and we use it as a baseline model to compare with other models that we propose. The spatial transformer network [38] is also discussed in this chapter. It learns a spatial transformer module along with the regular CNN to perform transformation operations conditional on individual data samples before classifying them. We show that such a framework is flawed as the proper spatial transformation operations cannot be perfectly recovered by a regular CNN.

In Chapter 3, we develop a mechanism to learn a deep neural network that can achieve transformation invariance using the student-teacher learning framework. The output from an intermediate layer of the teacher network is used as soft targets to guide the student network to produce the same feature representations even when the input is transformed. As a result, the feature maps extracted by the student network will be robust to the transformations. Moreover, only the shallow layers need to be trained to adapt to the transformed data while the downstream task-specific layers of the network do not need to be retrained or fine-tuned. Therefore, labeled data are not necessarily required for training.

A framework for training deep neural networks with optimal instantiations is proposed in Chapter 4. Latent variables are introduced to a regular deep architecture, separately for each class, to effectively remove nuisance transformations in the data and allow the model to obtain a reference pose for the object that is being classified. A two-step training mechanism is designed, which alternates between optimizing over the latent variables and the model parameters to minimize the loss function. We show our approach is able to obtain reference poses for the objects that are being classified. As a result, a support mask for a certain object class can be achieved by taking the average of the reference poses of objects from the same class, and they can be applied to decluttering images of objects with different types of random surrounding clutters. In addition, we show our approach can be extended to 3D volumetric data to recover the reference orientations of the 3D objects while classifying them.

Another direction that we explore in Chapter 5 is the use of 3D CAD models to render 2D images as a data augmentation approach. By generating 2D images from 3D CAD models, one can obtain a more compact way of representing an object class, which allows more transformation variations in data to be observed when training the models. Significant performance improvement is achieved by simply augmenting the real image training dataset with synthetic images, especially when the real image training dataset has a limited amount of examples. In addition, we show how a foreground-background segmentation model trained on synthetic images can be used to extract target objects from real images. This is particularly useful as it removes the nuisance factors of variations introduced by the background, and it allows a classification model trained on synthetic images with clear background to be directly applied to the real data.

CHAPTER 2

BASELINE MODELS

In this chapter, we evaluate two deep learning models that are designed particularly to deal with transformation variations in the data. In Section 2.1, we introduce the stacked statistical models with rotatable features proposed in [54], where a 3-layer layer-wised trained mixture model is introduced with the option of constructing an organized feature space that is equivariant to rotation or other cyclic transformations. In Section 2.2, we review the spatial transformer network introduced in [38], where a spatial transformer module is proposed and incorporated with a conventional CNN to disentangle nuisance factors of variations from the image objects. We present the experimental results of the both models to establish the baselines for the future chapters.

2.1 Statistical Models with Rotatable Features

We present a 3-layer feed-forward statistical model proposed in [54], which is a layer-wise trained mixture model with rotatable features. The model architecture contains an edge layer, a part layer, and an object layer for classification. As the model is layer-wise trained, labeled data is not necessarily required except for the classification layer. This model extends the part-based statistical object model of Bernstein and Amit [6] to incorporate rotatable part models and rotatable object models, and it allows the learned feature representations to be equivariant to the cyclic transformations in data. We describe the model architecture in the following.

2.1.1 Model Architecture

The 3-layer feed-forward architecture contains an edge layer, a rotatable part layer, and a rotatable object layer.

Edge Layer. The first layer is an edge detection layer that extracts binary oriented edge features at every pixel of a grayscale image. Based on the gray level edge detector described in [1], we extract eight binary oriented edges at each pixel based on the pixel intensity differences between the current pixel and the nearby pixels. For each detected edge for a pixel, a line of pixels within a certain radius oriented orthogonal to the edge orientation are defined as the neighborhood locations, and the extracted edge feature is spread to these neighborhood locations to make the edge detection robust to small variations. As a result, for an image $I \in \mathcal{I}$ where $\mathcal{I} \in [0,1]^{H\times W}$ with width of W and height of H, an edge feature map $x_I \in \{0,1\}^{H\times W\times 8}$ is produced by the edge layer.

Part Layer. The extracted edge features are passed to the part layer, where a dictionary of filters is learned from the edge patches in an unsupervised manner. During training, edge patches of size $h \times w$ with sufficient edge activities are selected from the edge feature maps. Let $x^{(n)} \in \{0,1\}^{h\times w\times 8}$ denote the edge feature map of the binary edge patch n. We assume the binary edge patches are samples of a mixture model with K mixture components where each component is a product of independent Bernoulli distributions. More specifically, we can model the edge patches as follows:

$$P(x) = \sum_{k=1}^{K} \pi_k \prod_{i=1}^{h\times w\times 8} \mu_{k,i}^{x_i}(1-\mu_{k,i})^{1-x_i} \tag{2.1}$$

where π_k is the mixing weights and μ_k is the mean parameters for the k-th mixture component. An EM algorithm can be designed to estimate the parameter values for each mixture component. After training, the learned mean parameters μ_k for each component can be used as the part models to create the part representations of the images.

In order to incorporate the rotation of the data into the model, an extension of the part layer is proposed in [54]. Instead of learning generic mixture components for all the sampled edge patches, we introduce a latent rotation variable to the mixture model and estimate rotation-equivariant mixture components. More specifically, let us denote

$x^{(n,1)}, \ldots, x^{(n,R)} \in \{0,1\}^{h\times w\times 8}$ as the R evenly spaced rotated versions of binary edge patch n. To achieve this, one can rotate an underlying image and extract features from the corresponding image patch to obtain rotated versions of a particular edge patch. One can stack the rotated versions of the edge patch to create a single binary feature vector $x^{(n)}$ and model this binary feature vector using a Bernoulli mixture model that has K mixture components with a latent rotation variable:

$$P(x) = \sum_{k=1}^{K} \pi_k \Bigg(\sum_{r=1}^{R} \pi_{r|k} \prod_{i=1}^{R\times h\times w\times 8} \mu_{k,\sigma_r(i)}^{x_i} \big(1-\mu_{k,\sigma_r(i)}\big)^{1-x_i} \Bigg) \tag{2.2}$$

where $\sum_{k=1}^{K} \pi_k = 1$ and $\sum_{r=1}^{R} \pi_{r|k} = 1$ for each k. σ_r is a cyclic permutation of indices that shifts the indices forward by $r\times h\times w\times 8$ entries to help align the rotation channels of the filter and the edge patch. The key step in this model is the cyclic permutation operation σ_r for indices of the rotated edge patches. This implies a structured organization of the parameter spaces of the mixture components, and it implicitly forces the part representations created by the part models to be equivariant to cyclic permutations. A modified EM algorithm is designed to estimate the parameters of the model with latent cyclic rotation. We derive the EM algorithm in Algorithm 1.

Using the learned dictionary of part models, one can create a part-representation of an image by encoding each pixel of the image by a binary feature vector. To achieve this, a neighborhood edge patch centered on the each pixel is extracted, and the i-th element of the binary feature vector is set to be one if this patch has highest likelihood under i-th part model. A max-pooling operation can be applied to make the feature maps robust to local translations.

Note that the rotatable mixture model we used in Equation 2.2 is a distribution over the combination of R rotated versions of an edge patch. Therefore, in order to create the part-representation of an image, ones needs to create and combine R rotated versions of the edge-representation of each neighborhood edge patch and calculate the likelihood of

Algorithm 1 EM Algorithm For Rotatable Part Model

```
1: procedure
2:     Denote $P_{k,r}(x^{(n)}) = \prod_{i=1}^{R\times h\times w\times 8} \mu_{k,\sigma_r(i)}^{x_i^{(n)}} \big(1-\mu_{k,\sigma_r(i)}\big)^{1-x_i^{(n)}}$ and $\pi_{r,k} = \pi_{r|k} \times \pi_k$.
3:     Choose an initial seting for the parameters $\theta^{(0)} = \{\pi_{r,k}^{(0)}, \mu_{k,i}^{(0)}\}$.
4:     $t \leftarrow 0$
5:     E step:
6:         $q_{r,k,n}^{(t+1)} = P(r,k|x^{(n)};\theta^{(t)}) = \frac{P_{k,r}(x^{(n)})\times\pi_{r,k}^{(t)}}{\sum_{r',k'} P_{k',r'}(x^{(n)})\times\pi_{r',k'}^{(t)}}$.
7:     M step:
8:         We solve for the following function:
9:             $\arg\max_\theta Q(\theta,\theta^{(t)}) = \sum_{r,k,n} q_{r,k,n}^{(t+1)} \big(\log P_{k,r}(x^{(n)}) + \log \pi_{r,k}\big)$
10:        Therefore, we have
11:            $\pi_{r,k}^{(t+1)} = \frac{\sum_n q_{r,k,n}^{(t+1)}}{N}$
12:            $\mu_{k,i}^{(t+1)} = \frac{\sum_{r,n} q_{r,k,n}^{(t+1)} x_{\sigma_r(i)}^{(n)}}{\sum_{r,n} q_{r,k,n}^{(t+1)}}$
13:    If the convergence criterion is not satisfied then
14:        $t \leftarrow t+1$
15:        and return to line 5.
16:    Stop
```

it under each part model. Obviously, this is computationally intensive and tedious. To resolve this, for each learned mixture component, we consider each part in every orientation a separate feature, i.e., we split the original mixture component into R sets of $h \times w \times 8$ entries, each becomes a separate part model. Therefore, we will have $K \times R$ part models, and each pixel of the part-representation of the image becomes a binary feature vector of size $K \times R$. Although each mixture component is broke up into separate part models for different rotations, we can still maintain the virtual associations for the part models within the same mixture component, and it allows the part-representations of the images to be rotation equivariant.

Object Layer. The part-feature maps from the part layer can be fed to the object layer, where a set of class-specific object models is learned for classification tasks. Training in this layer is supervised, where we learn object models for each class separately. The learning process in this layer is similar to that of the part layer that we described above except that this time we learn a model from the part-representation of the entire image instead of the edge-representation of randomly selected local patches.

Similarly, a rotatable object layer is also possible when we use the rotatable part layer as the second layer. When the part models are rotatable, any rotated version of part-feature maps can be created by transforming the part locations and then adjusting their angles. Therefore, for any part-representation of the data, we can create rotated versions and stack them together as a single binary part-feature vector. For each class c, M_c rotatable object models can be learned using the same kind of mixture distribution in Equation 2.2. After training, one can obtain the part-representation of a given test image by passing it through the first two layers. The part-representation of the test image is then fed to the object layer where the likelihood of it under each object model is calculated. The class label, as well as the orientation, can be determined by the object model that maximizes the data likelihood.

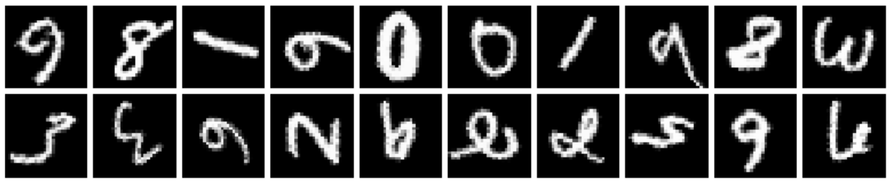


Figure 2.1: Examples of images from the mnist-rot dataset.

2.1.2 Model Performance

To establish a baseline for the future chapters, we apply the model to the mnist-rot dataset [46]. The mnist-rot dataset is a variation on the MNIST dataset where all the images are randomly rotated by angles uniformly generated between $0°$ to $360°$. Some examples of images from the mnist-rot dataset are shown in Figure 2.1. Classifying the images in this dataset is difficult as some rotated objects are easily mistaken for digits that come from a different class (e.g., the confusion between a digit 6 and a rotated digit 9. We will discuss other difficult cases in Chapter 4).

We apply our model to the mnist-rot dataset. For the rotatable part layer, we use 200 parts of size 6×6 with 16 rotations. The rotatable object models are used to calculate the likelihood for the part-representations of the images to perform the classification task. It shows good performance on the mnist-rot dataset, achieving an error rate of 5.76%. In Figure 2.2, we show a visualization of the learned rotatable object models trained with one mixture component per class. The first row and the second row show the visualization of the object models trained on the upright digits and the rotated digits respectively. Note that the preferred orientations of the object classes are not known to the model when training the rotatable object model. By allowing the learned rotatable object model to classify a few upright digits and figure out the preferable orientation, we can easily correct the orientations of learned object models, as is shown in the third row in Figure 2.2.

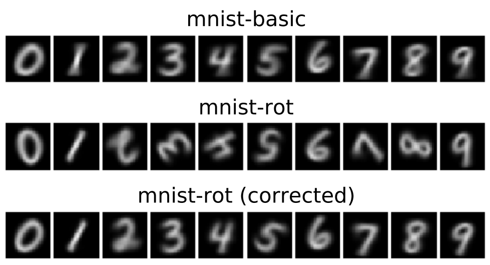


Figure 2.2: The first row and the second row show the visualization of the object models trained on the images with upright digits and randomly rotated digits respectively. The orientation-corrected object models are shown in the third row. Image source:[54].

2.1.3 Conclusion

In this section, we present a 3-layer feed-forward statistical model with rotatable features. Such a statistical framework enjoys some of the advantages of the existing deep learning architectures (including convolutional feature learning and weights sharing across spatial locations), but it offers a much simpler and more interpretable model.

In our experiment, we incorporate rotation into the model, and we show that the learned part models and the object models are explicitly rotatable. As a result, the output feature maps in the part layer and the object layer are equivariant to a discrete set of rotations. We note that this approach can generalize to any cyclic transformation, and the model can explicitly capture the transformation in data. The layer-wise training approach makes it possible to take advantage of the enormous amounts of unlabeled data when training for the first two layers and the class-annotated data is only required for training the object layer. During testing, if we conduct the likelihood-based test using the rotatable object models, not only the class label but also the orientation of the object can be determined by our model.

The simplicity and interpretability of the statistical model are appealing as it follows the statistical principle of likelihood and it is fast to train. It also shows strong results on variations of the MNIST dataset, achieving an error rate of 5.76% on the mnist-rot dataset. We will use this as a baseline model to compare with other models that we propose.

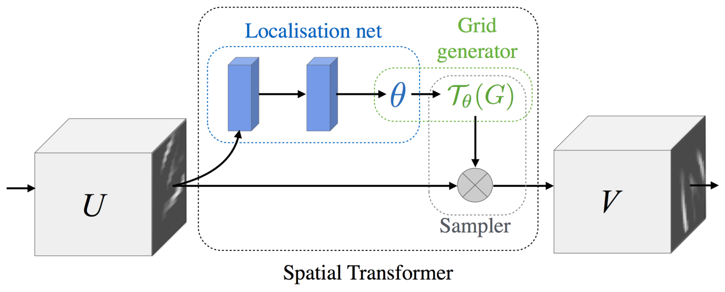


Figure 2.3: The architecture of a spatial transformer module. Image source:[38].

2.2 Spatial Transformer Network

Spatial transformer network [38] proposes a spatial transformer module that contains a localization network and a grid generator that can be trained to transform the input data. By combining it with a regular convolution neural network, the spatial transformer network can learn to not only actively transform the feature maps using spatial transformer module but also resolve the downstream task using the rest of the network at the same time.

2.2.1 Model Architecture

The key idea of the spatial transformer network is to learn a spatial transformer module that can perform transformation operations conditional on individual data samples before feeding the data samples to the regular convolutional neural network to handle a certain task. We show the architecture of a spatial transformer module combined with a regular CNN in Figure 2.3. A localization network is trained to regress the transformation parameters θ for a given input. Based on the transformation parameters, a sampling grid $\mathcal{T}_\theta(G)$ is then generated by transforming the regular spatial grid G. A warped output can then be produced by applying the sampling grid on the original input U, and it is further passed to the downstream network to handle a particular task. The designed transformation parametrization is differentiable with respect to the parameters, which allows the gradients

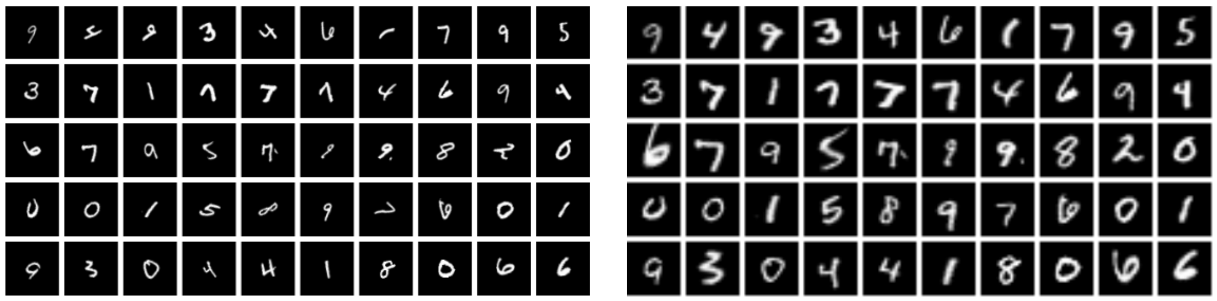


Figure 2.4: The spatial transformer module can rotate and scale the images to a canonical version of the images.

to be backpropagated to the localization network. Therefore, we can have an end-to-end training procedure for the spatial transformer network, and all the model parameters, including those of the spatial transformer module and the regular CNN, are trained under the supervision of task-specific signals.

What is surprising about the spatial transformer network is that the spatial transformation actions can be produced for the individual data samples, and this behavior is learned together with the training of the downstream task without additional supervision. This allows the network to capture the most task-relevant region of the image, and then transform it to a preferable form. Following the notation in Equation 1.1, we have the input $x = T_g\hat{x}$ is a spatially transformed version of a canonical version of the image $\hat{x}$. Therefore, the spatial transformer module is learning a function f that can recover the canonical form of the transformed image $T_g\hat{x}$, i.e., $f(T_g\hat{x}) = \hat{x}$.

In Figure 2.4, we show how the spatial transformer module recovers the orientations of some image examples. On the left, we show the test images of handwritten digits that have been randomly rotated, and the transformed (and subsampled) test images produced by the spatial transformer module are shown on the right. As we can see from the figure, although undesired rotations still exist in some examples, the orientations of the most images have been corrected, especially for those example images of digits of class one.

2.2.2 Discussion

Intuitively, the correctness of the recovered transformation operation for an image should be based on the class label of the image. As a result, the learned spatial transformer module should first recognize the class label of the current image, and then try to extract the transformation parameters by analyzing the pose of the image based on the class label.

To understand this, let us consider an example where the task is to classify images of rotated handwritten digits of class six and class nine. Note that although the rotated versions of the digits from these two classes have almost the same shape, they are still distinguishable due to some habits that people have in common when writing them. For example, the upper part of a digit six is usually more curved while a digit nine is often handwritten with a straight stem, and a digit nine is sometimes disambiguated by an underline. To correct the orientation of an image of rotated digit of class six, a spatial transformer needs to recognize it as a digit of class six rather than a digit of class nine before producing the transformation parameters. Otherwise, it is possible that a different transformation would be produced to rotate the image to look like an image of digit of class nine. There are some more such examples that will be discussed in Chapter 5. Therefore, we argue that sometimes the proper spatial transformation is undefined, and an accurate estimate of transformation is not possible unless the image label is captured.

However, in the spatial transformer network, we do not have separate spatial transformation modules for examples from separate classes. Instead, only one spatial transformer network is learned to produce the transformation actions for all images without knowing the labels of the images. Since the localization network itself has the architecture of a regular CNN, it is not invariant or equivariant to transformations such as rotation and deformation. It raises a question of whether the spatial transformer module distinguishes the class labels of the images before transforming them. In fact, it is not surprising that the spatial transformer module does not contain much information about the class labels. Otherwise,

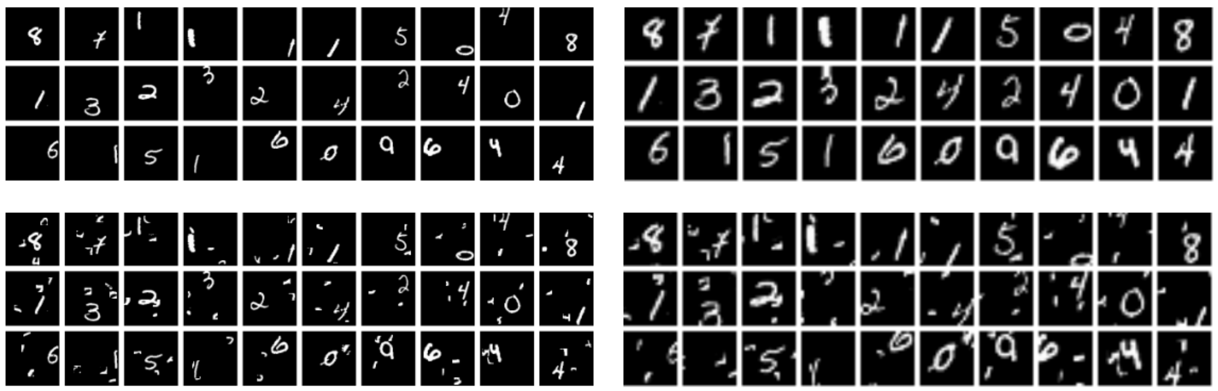


Figure 2.5: The spatial transformer module is sensitive to random clutter if they do not exist in the training data.

we would not need the downstream CNN network to handle classification task.

2.2.3 Experiment

We design several experiments to investigate the properties of the spatial transformer network and evaluate the performance of the model.

In order to verify if the spatial transformer recognizes the class labels before classifying the images, we conduct the following experiment. We train a spatial transformer network using images of shifted objects and take the output of the second last fully connected layer (the layer before the final layer that produces transformation parameters) in the localization network as the feature representations for each image. Then we train a SVM classifier to classify an image based on the feature representations of that image. We find that the trained SVM classifier has a poor classification performance that is barely better than a random guess. This indicates that the fully connected layer in the localization network does not contain much information about the class labels. A possible conjecture is that the localization network estimates the location of the pixel intensity blob and its rough orientation and scale, and the transformation matrix is produced based on that. To verify this, we randomly add some clutter to the test images and apply the spatial transformer module to generate transformation operations for the images. Note that such random clutter is not observed in the training data, and the spatial transformer network is trained on images with a clear background. In Figure 2.5, we show that the spatial transformer module is sensitive to random clutter in the background if such clutter does not exist in the training data, and it is difficult for the spatial transformer module to recover the orientations of the image objects when such random clutter exists in the background.

On a separate note, we find that a spatial transformer module can be explicitly trained to regress the transformation parameters when the target values of the transformation parameters are provided as the supervised signals during the training.

To establish a baseline for the future chapters, we train a spatial transformer network on the mnist-rot training dataset [46] and apply the model to classifying the rotated images. We achieve an error rate of 5.71% on the mnist-rot test dataset.

2.2.4 Conclusion

The idea of learning to recover the reference poses of the images together with the downstream tasks proposed in the spatial transformer network is appealing. Ideally, the nuisance factors of variations such as rotation, shifting, and deformation can be decoupled from the task-specific networks, which would make the problem easier for the task-specific networks. The self-contained spatial transformer module can be easily combined with a regular CNN, and it can be trained in an end-to-end fashion.

However, as we discussed above, such a framework is flawed, as the proper spatial transformation cannot be perfectly recovered by a regular CNN. We show that, in some cases, a proper reference pose of an image is difficult to estimate or even undefined if the class label of the image is not recognized first.

Therefore, we argue that architectural changes to the existing deep learning models are necessary if one wants to recover the reference poses of the images. In the next few chapters, we propose different approaches to deal with transformation variations in the data.

CHAPTER 3
TRANSFORMATION-INVARIANCE VIA STUDENT TEACHER LEARNING

3.1 Introduction

As we mentioned above, the most common and convenient way to achieve the transformation invariance for the various models is through the use of data augmentation. Different transformed versions of the original data can be collected and used for training the models. The more variations of the data are observed, the models will become more transformation invariant. By augmenting the training data using different approaches [20, 45, 18, 17, 77], it has been shown that such methods can achieve significant performance improvement.

Despite the success of this approach, it has two drawbacks. First, such an approach usually requires us to either create or collect the transformed data from some existing labeled dataset. When a certain transformed version of the data is difficult to create or not readily available in the existing labeled dataset, we are no longer able to adopt such approaches. Second, it is usually difficult for such an approach to take advantage of unlabeled data, which contains an enormous amount of variations that could be useful for vision tasks.

In this chapter, we propose a procedure to learn a deep neural network that could achieve transformation invariance and take advantage of training examples that are not necessarily labeled. We call a learned network transformation invariant if the network can produce the same feature maps even though the input has been transformed. To achieve this goal, we adopt the student-teacher learning framework [4, 36] for our model and pairs of two unlabeled training images, one canonical image and one transformed version of the same image, are provided for training the student network. Given a pre-trained teacher network, we use the mid-layer representations of the canonical input as the supervisory signals, and the student network is trained to reproduce the same mid-layer representations when operating on the transformed version of the input. As a result, the learned feature maps will be robust to transformations. Moreover, only the shallow layers need to be trained to adapt to the transformed data while the downstream task-specific layers of the network do not need to be retrained or fine-tuned. Therefore, labeled data are not required for training the student network.

3.2 Related work

Recently, a model distilling approach was proposed to compress multiple pre-trained models into a single one [4, 36] or to perform knowledge transfer from a (complex) model to another (simple) one [65]. By using a student-teacher learning mechanism, training of a student model is guided by the soft targets produced by a teacher model. The idea is to train the student network to capture not only the class information provided by the labels but also the finer structure learned by the teacher network.

Besides the successful application on compressing the pre-trained models, this training mechanism has also been used for supervision transfer tasks [26]. A student model can be trained for vision tasks where the images have a different image modality. One can train the student model to reproduce the mid-level semantic representations learned from a well-labeled image modality for modalities for which there are paired images and less well labeled (e.g., depth images, optical flow images, etc.). This allows the model to use the paired images of the two modalities and utilize the mid-level representations from the labeled modality to guide the representation learning on the other unlabeled modalities.

In this project, we are inspired by the student-teacher learning mechanism and apply it to the context of transformation-invariant feature learning. We use the mid-level representations from a teacher network as the supervisory signals, and teach the student network to reproduce the feature maps when the input is a transformed version of the same image.

3.3 Method

A state-of-the-art deep learning model has already done an excellent job finding features that are useful for a vision task. However, the features learned by such deep neural nets are sensitive to transformations that were not observed in the training data, and the network might produce a different feature map if we transform the image. Therefore, our intuition is to build a deep learning model to produce the same feature maps even if the input is transformed. The mechanism of learning such models is presented in Figure 3.1.

More specifically, as we can see from the figure, the model architecture contains a combination of two networks: A teacher network that gives the state-of-the-art performance for the original dataset; and a student network that tries to "de-transform" the feature maps of the transformed version of the data. By doing this, the feature maps generated for the transformed data and the canonical data by the student network and the teacher network become close. To achieve this goal, we match the feature maps of two networks by regressing the outputs of the matching layers using L_2 loss, i.e., the training of the student network is guided by the soft targets produced by the teacher network at a particular layer. When the soft targets have high entropy, it usually provides more information than a simple hard target. Note that no label information is required during training as the model only learns to regress the output of the intermediate feature maps. After training, the student network can append the downstream task-specific layers from the teacher network to perform a given task without retraining.

Training such a model requires a two-step approach: we first train a state-of-the-art deep learning model or take an existing model that gives a good performance on the given task. We freeze the weights of the model and refer to it as the teacher network. Then we build a d-layer student network that has the same architecture as the top d layers of the teacher network. During training, a canonical version of the input is presented to the teacher network, and the feature map of the image can be extracted at the d-th layer. Meanwhile,

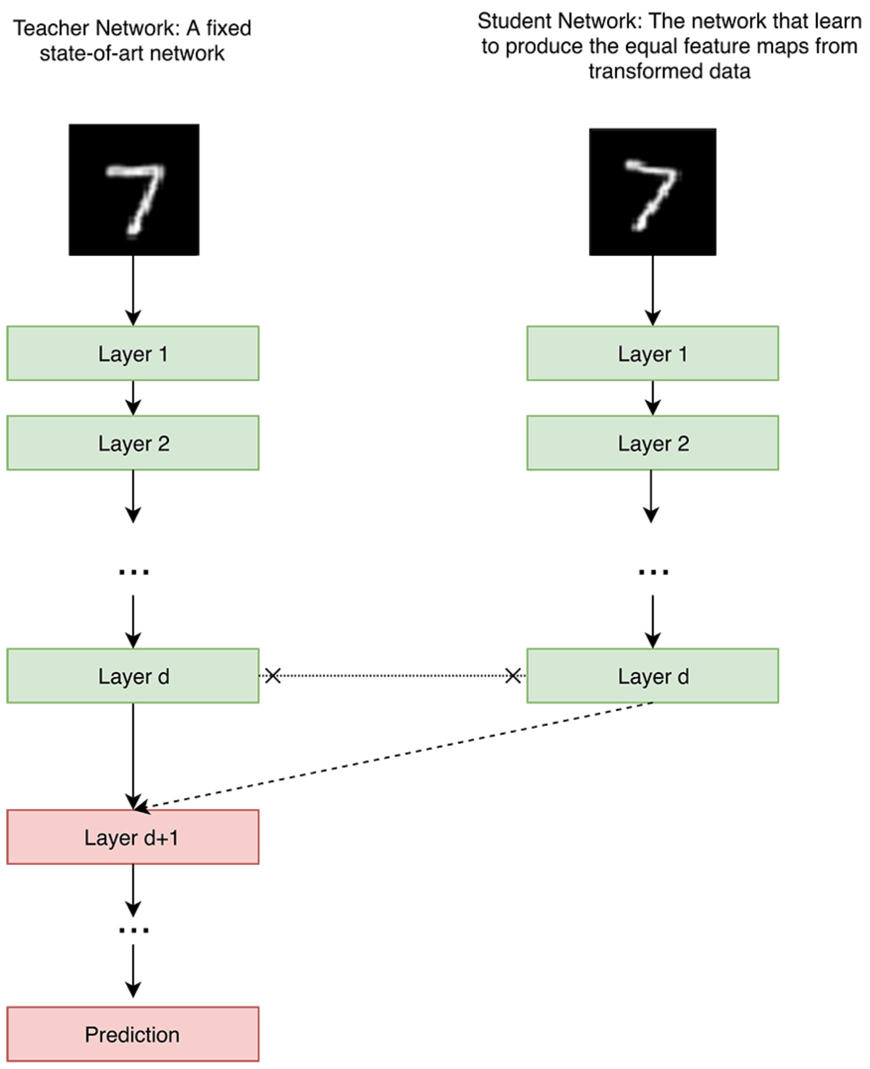

Teacher Network: A fixed state-of-art network
Student Network: The network that learn to produce the equal feature maps from transformed data
Layer 1
Layer 2
...
Layer d
Layer 1
Layer 2
...
Layer d
Layer d+1
...
Prediction

Figure 3.1: We show the joint architecture of a transformation-invariant deep neural net by student-teacher learning. The left one is the teacher network that achieves state of the art for a given task, and the right one is a student network that needs to be trained. During training, the teacher network will take a canonical image as the input while a transformed version of the same image will be created and passed to the student network. Guided by the supervisory signal from the output from the d-th layer of the teacher network, the student network learns to reproduce the same feature maps by regression. After training, the student network will append the learned d layers with the downstream layers of the teacher network ($d + 1 \ldots, n$ layers) to perform a given task.

a transformed version of the input is presented to the student network, and it will take the feature outputs from the d-th layer of the teacher network as the soft targets for regression. In our experiment, we simply use the Euclidean distance between the desired feature output and the generated feature output as the loss function.

During testing, the top d layers of the student network will be connected to the downstream layers of the teacher network to perform a certain task. As the student network learns to reproduce the output of the teacher network and "de-transform" the feature maps for a transformed input, the downstream layers of the teacher network are not required to be further fine-tuned or retrained for the task.

A natural choice of the matching layer would be the last layer, where the class probabilities are produced by the original model. The student network will learn to produce the same class probabilities when a transformed version of the input is given. It is worth noting that it only makes sense to use the class probabilities as the soft targets when the transfer dataset consists of the data from the same set of classes. If the transfer dataset contains unlabeled data from a different dataset and we want to learn the transformation-invariant filters for local patterns, it would be more appropriate to use the feature maps on the shallower layers as the soft targets. The choice of matching layer depends on the level of transformation invariance we want to learn. We will explain this more in detail in the next section.

3.4 Image Deformation

We want to show that the student-teacher learning mechanism can help achieve invariant representations for different types of transformations, including rotations, translations, as well as random deformations that are not well defined. The rotated version and the translated version of an image are usually easy to acquire, but it is unclear how to perform random deformations for an image. In this section, we adopt the framework of image deformation parameterization introduced in [1], and we show how to generate randomly deformed images by generating smooth random deformation functions. To do so, let us first review some key concepts of the wavelet basis functions, and we will see how this can play a part in generating deformed images.

3.4.1 Function Parametrization by Wavelet Bases

An wavelet basis is an orthonormal bases of functions that can be used to parametrize functions, and it is especially suited for a coarse-to-fine function parametrization.

For example, let us first consider the 1D wavelets on the unit interval. Basis functions of a 1D wavelet with S levels can be organized hierarchically with $2^s - 1$ functions at each level, where $s = 1, 2, \ldots S$. More specifically, at a given level s, a wavelet basis function indexed by $l = 0, \ldots, 2^{s-1} - 1$ as the shifting factor on the unit interval can be written as:

$$\psi_{s,l}(x) = \psi_{s,0}(x - 2^{-(s-1)}l) \tag{3.1}$$

and we have:

$$\psi_{s,0}(x) = 2^{(s-S)/2}\psi_{S,0}(2^{(s-S)}x) \tag{3.2}$$

where $\psi_{S,0}(\cdot)$ is called the mother wavelet. As we can see, all the basis functions can be obtained by scaling, dilution, and shifting of the mother wavelet. As the level increases, both the scaling on the mother wavelet and the possible values of l will increase, and it leads to a finer resolution of the parametrization.

For 2D wavelets, the basis functions are still organized hierarchically with multiple levels. To index the wavelet basis functions, in addition to the level parameter s and the shifting factor l that are used in the 1D wavelets, an extra index α is introduced in the 2D wavelet where $\alpha = 1, 2, 3$ is used to indicate the horizontal, vertical and diagonal detail components. More specifically, a 2D wavelet basis function on the unit square at level s can be written as follows:

$$\psi_{\alpha,s,l_1,l_2}(x) = \psi_{\alpha,s,0,0}(x_1 - 2^{-(s-1)}l_1, x_2 - 2^{-(s-1)}l_2) \tag{3.3}$$

where $\alpha = 1, 2, 3$ and $l_1, l_2 = 0, \ldots, 2^{s-1} - 1$. Similarly, ψ_{α,s,l_1,l_2} is a scaling, dilution and shifting of the mother wavelet $\psi_{\alpha,S,0,0}$, and we have:

$$\psi_{\alpha,s,0,0}(x) = 2^{(s-S)}\psi_{\alpha,S,0,0}(2^{(s-S)}x) \tag{3.4}$$

In the 1D and 2D wavelet bases, the basis functions with low levels are smooth functions that describe global variations whereas the basis functions with high levels describe more local variations. Therefore, such a hierarchical organization of the basis functions is naturally suited for coarse-to-fine parametrization of functions. In the next section, we will describe how we can take advantage of this property and use the 2D wavelet basis functions to parametrize image deformations. Note that a particular form of the mother wavelet ($\psi_{S,0}$ for the 1D wavelet and $\psi_{\alpha,S,0,0}$ for the 2D wavelet) needs to be carefully chosen in order to make the basis functions orthonormal. In this work, we use the family of Daubechies wavelets as the basis functions for parametrizing the functions.

3.4.2 Create Deformed Images

Let us consider the image domain D as a continuum. An original image can be defined on the domain D by $F(x^s)$, where $x^s \in D$. One can introduce a smooth deformation function ϕ, mapping from D to D. Therefore, a deformed image associated with the deformation function ϕ can be expressed as $\tilde{F}(x^t) = F(x^s) = F(\phi(x^t))$, where $x^t \in D$. In this case, for any point in the original image F, ϕ^{-1} will deform it to the corresponding position in the image $\tilde{F}$.

To make it easier to work with, we define a displacement field as $U(x) = \phi(x) - x$. When $U(x) = 0$ for $\forall x \in D$, then $\phi(\cdot)$ is an identity map. As described in [1], in order to preserve

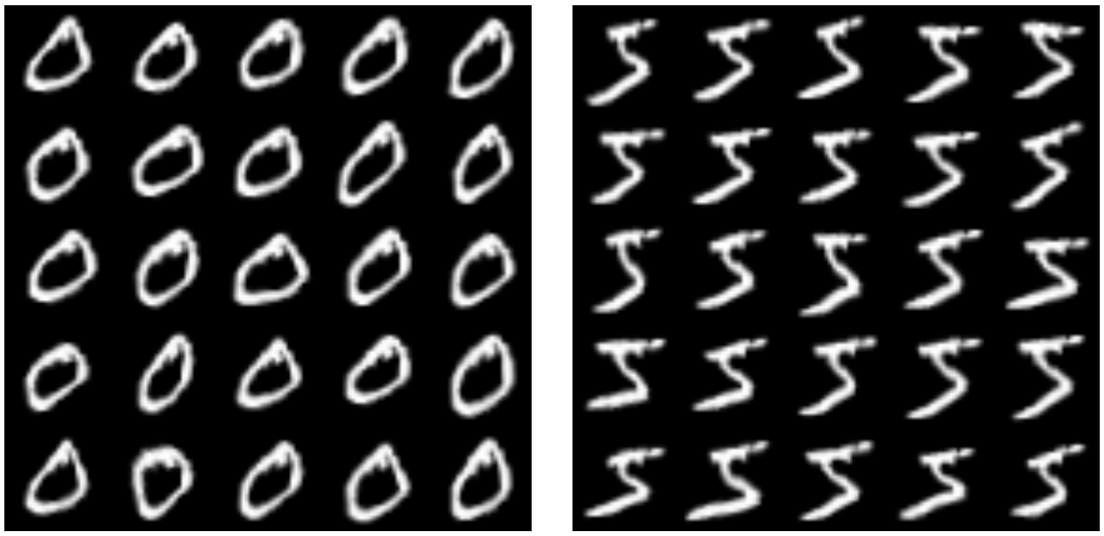


Figure 3.2: Visualization of some randomly-deformed training examples from the MNIST dataset. Although some parts of the class object are distorted, they look like different instantiations of an object coming from the same class.

the topography of the image after deformation and introduce a smooth deformation function, one can use wavelet basis functions to obtain a natural coarse-to-fine parameterization of the deformations. Therefore, denote $U^{(1)}, U^{(2)}$ as the two components in the 2d displacement field, we have the following:

$$U^{(q)}(x) = \sum_{k=0}^{d} u_k^{(q)} \psi_k(x), q = 1, 2 \tag{3.5}$$

for some finite d, and ψ_k is the 2D wavelet basis function. Here $k = (\alpha, s, l_1, l_2)$ is a four-parameter index for the 2D wavelet basis function. Note that we have the 2D wavelet basis function $\psi_{\alpha,s,l_1,l_2}(x)$ defined in Equation 3.3, where $\alpha = 1, 2, 3$ represents the horizontal, vertical and diagonal detail components respectively, and $l_1, l_2 = 0, \ldots, 2^{s-1} - 1$ represent the shifting factor. We can thus parametrize the random displacement field by setting the coefficients of the basis functions with values that are independently Gaussian distributed with mean 0 and variance $1/\lambda_s$. λ_s will change according to s, which is set to maintain the smoothness of the function. Note that as s increases, the basis function focuses more

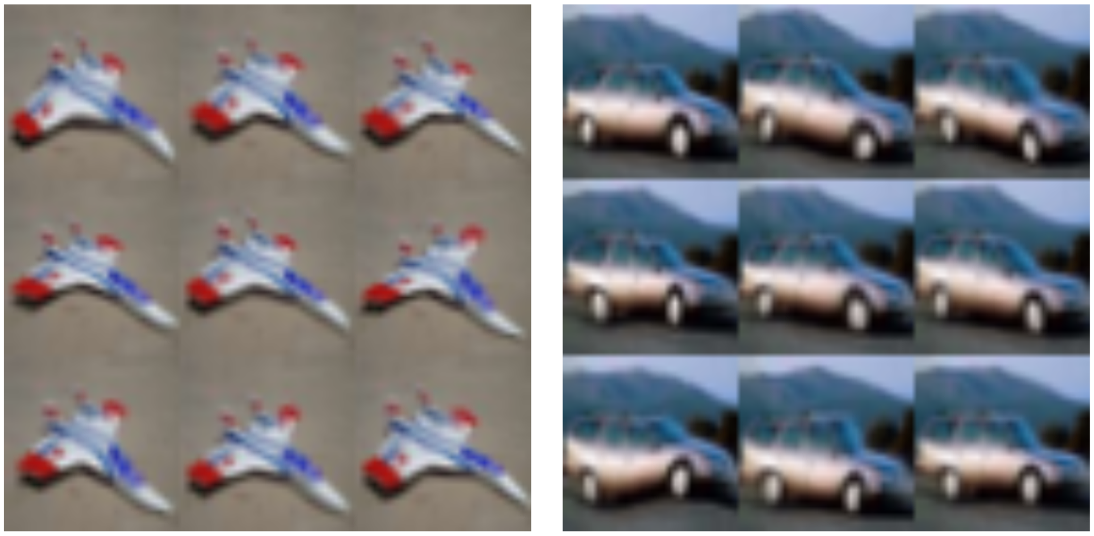


Figure 3.3: Visualization of some randomly-deformed training examples from the CIFAR-10 dataset. Although some parts of the image object are distorted, they look like different instantiations of an object coming from the same class.

on the local regions with smaller supports. The basis functions with low levels are smooth functions that describe global variations. In order to enforce the smoothness of the random displacement field, we impose $\lambda_s = 2^{\rho s}$ so that the variance of the coefficients would descrease as s increases. In this way, the displacement field represented by Equation 3.5 is smooth.

In order to make the random displacement field more smooth, we further set the coefficients of the basis functions with high levels as zero, and randomly generate coefficients with Gaussian distribution with zero mean and variance of $1/\lambda_s$ for basis functions with low levels. After that, we can get the pixel values for the deformed images by referencing the displacement field map and bilinear interpolation. We show the randomly deformed images from the MNIST dataset and the CIFAR-10 dataset in Figure 3.2 and Figure 3.3. For the deformed handwritten image in Figure 3.2, we show examples of images deformed from an original image of digit zero and an image of digit five. As we can see, these images are very similar to those in the original dataset, and they are essentially different instantiations of the same image. It would be valuable to find some feature representations that are invariant to deformations like these. Deformations on the natural images make the images less realistic in Figure 3.3, but they can still be easily recognized as the different instantiations of the same image.

3.5 Experiment

We conduct several experiments on the MNIST dataset and the CIFAR-10 dataset using the student-teacher training mechanism and compare our results to the methods that do not use such a training mechanism. In order to eliminate the possible transformation invariance learned from the original dataset, we limit the number of training examples for training the teacher network. We use the MNIST1000 (first 100 images per class from the MNIST training dataset) and the CIFAR-10(10k) (first 1000 images per class from the CIFAR-10 training dataset). For the MNIST1000 dataset, the architecture we select for the teacher network achieves an average error rate of 3.17% on the original test set, and we achieve error rates of 19.12% for the CIFAR-10(10k) dataset using our selected architecture.

In this section, we experiment with the student-teacher training approach and see how well it can handle the data with rotations and random deformations. We also show how it can be applied to the unlabeled data.

3.5.1 Training with Pairs of Rotated Images

To evaluate the model performance on a transformed dataset, we create a rotated test dataset for each of the datasets: for each image in the test dataset, we create five different rotated images with randomly selected degrees from -20° to -5° or from 5° to 20°. If no augmented data is allowed for training, then the teacher network trained on the original MNIST1000 dataset achieves an average error rate of 6.4% on the rotated test set and the teacher network trained on the original CIFAR-10(10k) dataset achieves an average error rate of 25.72% on the rotated test set.

We then start to experiment with the student-teacher training approach as well as the

Table 3.1: Experiment Result on MNIST1000 with rotation (100 images per class; 5 tries for each experiment)

Model Description	Result		
Original Model (test on original test set)	96.83%		
Original Model (test on rotated test set)	93.60%		
Model trained with augmented data (test on original test set)	97.52%		
Model trained with augmented data (test on rotated test set)	96.81%		
Trained Student Network on Different Layer as Soft Target	Last Conv Layer	First Fully Connected Layer	Second Fully Connected Layer (before softmax)
Student Network trained on MNIST1000 (test on original test set)	96.97%	97.54%	97.72%
Student Network trained on MNIST1000 (test on rotated test set)	94.41%	96.57%	96.92%

data augmentation approach. When training the student network, we first determine which layer of the teacher network is used as the matching layer. During training, a pair of images will be presented to the teacher network and the student network separately. The teacher network will take the canonical version of the input and produce the feature maps at the matching layer as the supervisory signal for the student network. A random rotation with a degree from -20° to 20° is applied to the same input image, and the rotated image is passed to the student network as the input. An $L2$ loss is used to guide the student network to produce the same feature maps for the matching layer even though the input is randomly rotated. To compare with the traditional data augmentation approach, we also train a CNN on an augmented dataset. We augment the original training data by randomly rotating the images with a degree from -20° to 20°, and the amounts of the training data for the both approaches are the same.

How much do we gain if we adopt the student-teacher training mechanism, compared to the traditional training approach and other simple transformation-invariance training approaches such as the data augmentation approach? In Table 3.1 and Table 3.2, we show

Table 3.2: Experiment Result on CIFAR-10(10k) (1000 images per class)

Model Description	Result		
Original Model (test on original test set)	80.88%		
Original Model (test on rotated test set)	74.28%		
Model trained with augmented data (test on original test set)	81.60%		
Model trained with augmented data (test on rotated test set)	78.00%		
Trained Student Network on Different Layer as Soft Target	Last Conv Layer	First Fully Connected Layer	Second Fully Connected Layer (before softmax)
Student Network trained on CIFAR-10(10k) (test on original test set)	75.04%	79.91%	81.40%
Student Network trained on CIFAR-10(10k) (test on rotated test set)	76.29%	80.34%	80.71%

results of experiments on the MNIST1000 dataset and the CIFAR-10(10k). We observe that both the data augmentation approach and the student-teacher training approach can improve the performance of the original model on the original test set and the rotated test set. The student-teacher training mechanism achieves comparable performance with the data augmentation approach. Especially when tested on the rotated test set, the student-teacher training mechanism achieves a slightly higher accuracy on both datasets. We also show that the deeper layer we use as the soft target, the better accuracy the model achieves. This is expected because more class information is conveyed by each neuron in a deeper hidden layer. We note that the choice of the matching layer depends on the level of transformation invariance we want the model to learn. Unlike the units in the fully connected layers, a unit in the last convolutional layer only extracts a pattern for a local receptive field. Therefore, we can only learn transformation invariance in a local region if we use the last convolutional layer as the matching layer.

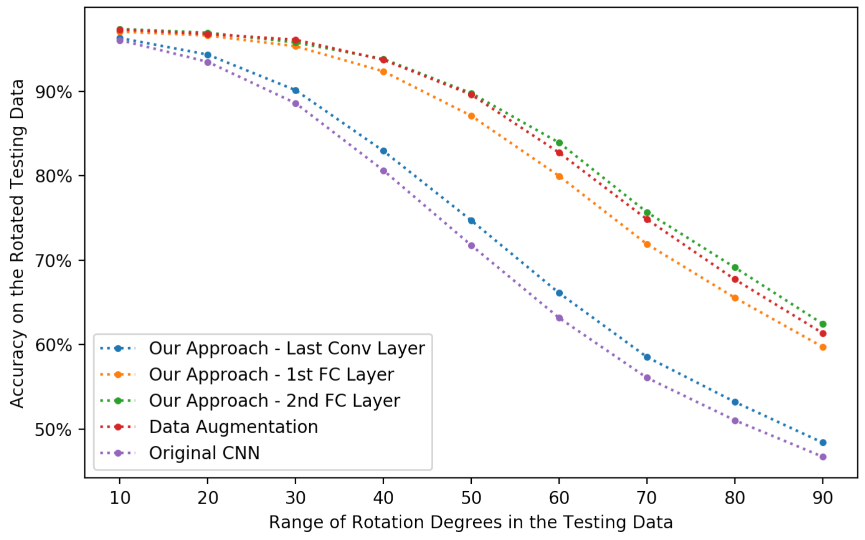


Figure 3.4: Classification accuracy rates by different methods when tested on the MNIST test datasets that contain different ranges of random rotation. Each model is trained on the MNIST1000 training data augmented by -20° to 20° random rotations.

As is shown above, both the data augmentation approach and the student-teacher training approach can improve the performance of the original model on the original test set and rotated test set. One interesting question is that when the models are trained on a dataset of images that are rotated by degrees within a certain range, how does the performance change when the testing data has a wider or narrower range of random rotations? To answer this question, we train the models on a fixed training dataset of rotated images (where the images are randomly rotated with degrees from -20° to 20°), and we want to explore the performance on the test datasets that contain different ranges of random rotations. In Figure 3.4, we observe that the performance of all approaches drops dramatically when the test images have a wider range of random rotation. The performance degradation of the student-teacher training approach and the data augmentation approach are similar, and both of these two approaches outperform other approaches by a big margin. We also show the accuracy result by a CNN baseline model that is trained on the MNIST1000 dataset without any data augmentation. As we expected, the conventional CNN trained on the original dataset is not able to observe much variations in the data and therefore produces the worst results among the approaches we proposed.

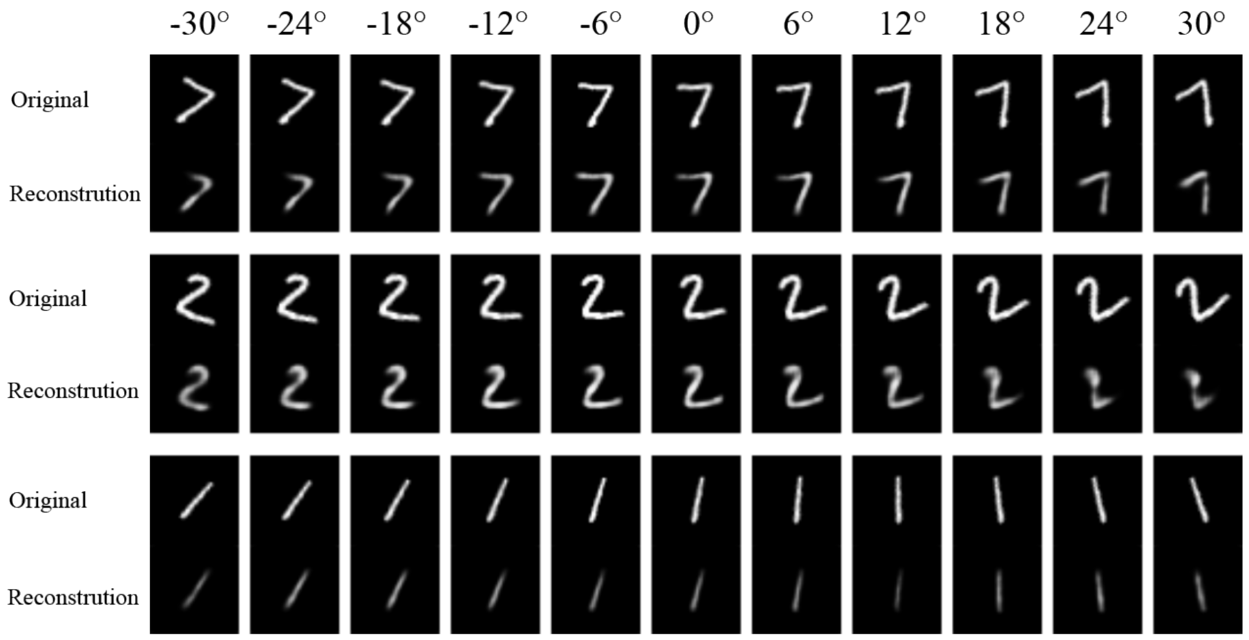


Figure 3.5: Visualization of the reconstructed image by the student network.

In order to show that the student network learns to "unrotate" the feature map, we use the teacher network as the encoder network and learn a decoder for the teacher network to reconstruct the input images. After learning the decoder for the teacher network, we connect the student network with the learned decoder and construct a new autoencoder for image reconstruction. As the student network tries to produce the same bottleneck representations regardless of the rotation of an image, ideally the reconstruction of a rotated image will be an upright version of the image. In Figure 3.5, we show some examples of reconstructed images when rotations between $-30°$ to $30°$ are applied to the input images. Although the reconstructed images are not exactly the upright versions of the original image, we observe that the student network still manages to correct some rotations in the images, especially for those rotated versions that are close to the original input image.

3.5.2 Training with Pairs of Randomly Deformed Images

To show that the student-teacher learning mechanism can learn representations that are robust to other types of transformation, we want to apply our approaches to images with random deformations. Following the procedure we introduced in Section 3.4, we create a deformed MNIST test dataset: for each image in the MNIST test dataset, we create five different deformed images. Using an original CNN that trained on the MNIST1000 dataset, the model makes an average of 4.65% test error on the deformed test set.

We then train the model using the data augmentation approach and the student-teacher approach. Similar to the experiment above, when training the student network, we present a canonical image to the teacher network for feature extraction and a deformed version of the same image for training the student network to produce the same feature maps. We also train a regular CNN using the data augmentation approach, where the data is augmented with images that are randomly deformed. The amounts of the training data we use for the both approaches are the same.

We show the experiment results in Table 3.3. The data augmentation approach and the student-teacher training approach further improve the result on the MNIST1000 dataset, and both achieve error rates that are close to 2%. We also observe that the student network

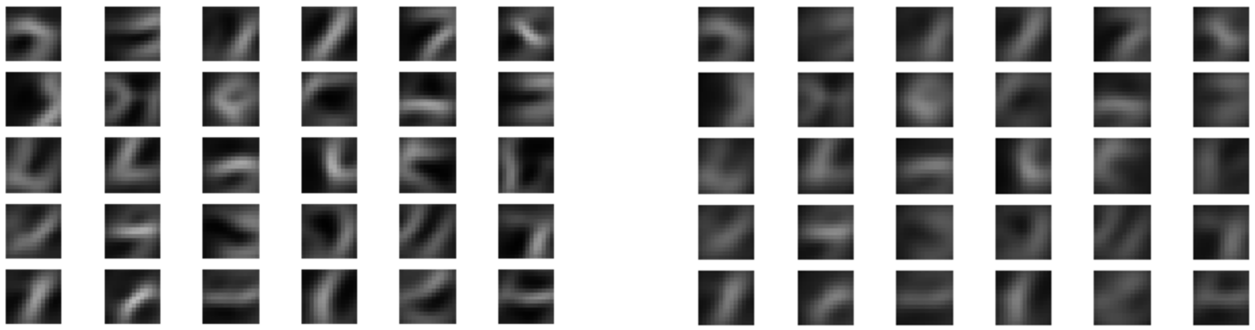


Figure 3.6: Visualization of the filters learned by the teacher network and student network when choosing the last convolutional layer as the soft target. The left are the visualization of the filters bank learned by the teacher network, and the right are the visualization of the filters bank learned by the student network.

Table 3.3: Experiment Result on MNIST1000 with deformation (100 images per class; 5 tries for each experiment)

Model Description	Result		
Original Model (test on original test set)	96.83%		
Original Model (test on deformed test set)	95.35%		
Model trained with augmented data (test on original test set)	97.98%		
Model trained with augmented data (test on deformed test set)	97.25%		
Trained Student Network on Different Layer as Soft Target	Last Conv Layer	First Fully Connected Layer	Second Fully Connected Layer (before softmax)
Student Network trained on MNIST1000 (test on original test set)	97.03%	97.69%	98.01%
Student Network trained on MNIST1000 (test on deformed test set)	95.72%	97.02%	97.40%

achieves better accuracy performance when we use the deeper layer as the soft targets, which is consistent with what we observed in the previous experiments.

We compare the convolution filters learned in the student network and the teacher network. Let us consider the case where we use the feature maps of the last convolutional layer as the soft target. In Figure 3.6, we show the visualization of the filters learned by the two networks when the MNIST1000 dataset is used as the transfer dataset. The visualizations are obtained by taking the weighted average of image regions that have positive activations on the neurons. The filter visualizations from the student network on the right have similar but much blurrier patterns comparing to those learned by the teacher network. For a particular local pattern that activates a unit in the teacher network, we expect a corresponding unit in the student network should also be triggered by a transformed version of this pattern. Therefore each unit in the student network detects patterns that have a wider range of variations, yielding a much blurry visualization.

3.5.3 Training with Pairs of Unlabeled Data

One of the advantages of the student-teacher training mechanism is that it can be trained with unlabeled data. We create an unlabeled dataset where each unlabeled example is generated by randomly sampling a 10×10 subregion from the original images in the MNIST1000 dataset and putting this patch at a random location of a blank image with a black background (see some examples in Figure 3.7). We use the generated unlabeled data as the transfer dataset to train the student network. Since the generated images contain only the local regions of the original images, it helps the student network to learn the transformation invariance for local patterns.

To show the advantage of using the unlabeled data, we conduct the following experiment: We train a regular CNN model using an augmented MNIST1000 dataset where images are randomly rotated during training, and we achieve an accuracy rate of 98.58% on the rotated test images of digits of class one. The model performance drops to 96.75% if we train the CNN model using the augmented MNIST1000 dataset where all images are randomly rotated except those from digit class one (images of the canonical version of digits of class one are still shown to the model). However, if we train a student network with the unlabeled data that contains 10×10 subregions sampled from images of all classes except digit one, the network

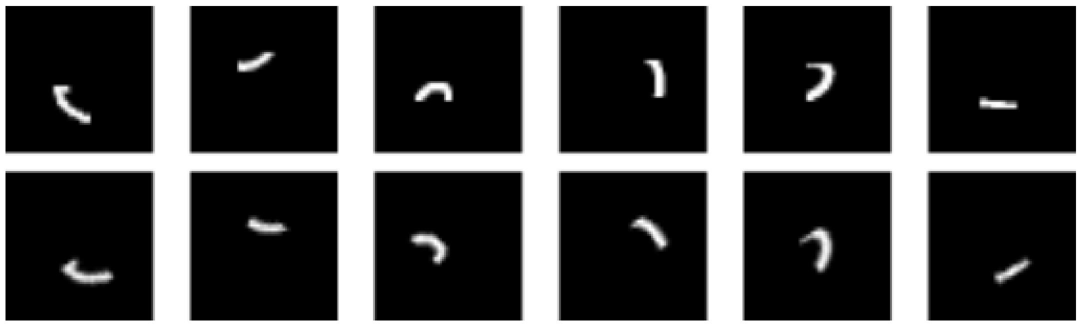


Figure 3.7: Five pairs of generated unlabeled images. Top row shows images that contain 10×10 subregions sampled from the MNIST1000 dataset. Bottom row shows rotated versions of images in the top row.

still has an average of 97.52% test accuracy on test images of rotated digits of class one. It is only 0.2% worse than the performance of a student network trained on the unlabeled images of subregions sampled from all classes. It is possible that the transformation-invariant representations learned from subregions of images from other digit classes (e.g., digit seven and digit nine) are sufficient, so that the student network does not need to see examples from digits of class one to "unrotate" the feature maps for rotated digit ones.

In addition to that, we can also use a different dataset where the class labels are different from those of the transfer dataset. For example, for the CIFAR-10 dataset experiment, we can use images from the CIFAR-100 dataset as the unlabeled data. During training, a regular CNN pre-trained on the CIFAR-10 dataset is used as the teacher network, and the student network is trained to match the feature maps of the CIFAR-100 images produced by the teacher network. We observe that the accuracy rate on the rotated CIFAR-10 test set improves to 75.91% if we use the CIFAR-100 dataset as the transfer dataset, compared to an accuracy rate of 74.28% achieved by the original CNN model.

3.6 Conclusion

In this work, we adopt the student-teacher training mechanism to learn a deep neural network that is invariant to general types of transformations. Using the output from an intermediate layer as the soft targets not only enables the trained student network to remain robust to the task-irrelevant transformations but also provides a way to use enormous amounts of unlabeled data to provide such robustness. Especially when the image transformation is not well defined or producing an augmented dataset that has such transformation is not possible, our mechanism can be helpful as long as a training set with paired training examples (each pair contains the canonical form and the transformed version of an example) is provided. For example, such paired training examples can be extracted from image sequences of video clips, or paired images taken by cameras in different positions, etc.

We were not aware of the work on cross-modal distillation by Gupta et al.[26], which has a similar network architecture with our proposed model. However, their work mainly focuses on training a CNN model for a new image modality by teaching it to reproduce the same mid-level representations from a well-labeled image modality when operating on the same image objects. In our work, we mainly investigate whether this kind of student-teacher learning mechanism allows the learned network to be robust to local transformations and produce the same mid-level representations even when the images are transformed.

CHAPTER 4
DEEP LEARNING WITH OPTIMAL INSTANTIATIONS

4.1 Abstract

One of the challenges for object recognition is the extraneous variations in the data, such as shifting, rotation, and deformation. In this paper, we design a framework for training deep neural networks with optimal instantiations, where latent variables are introduced, separately for each class, to effectively remove nuisance transformations in the data and allow the model to obtain a reference pose for the object that is being classified. We apply a two-step training mechanism for our framework, which alternatively optimizes over the latent variables and the model parameters to minimize the loss function. We show that CNNs trained using our framework 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 trained on subsets of training data.

4.2 Introduction

In recent years, machine learning algorithms that involve deep architectures have gained popularity. The expressiveness of deep 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 deep neural networks, as the traditional networks without special design do not generalize to unobserved variations in the data. It would become much easier for downstream tasks if such task-irrelevant variabilities can be removed from the data.

In order to learn transformation-invariant representations, many have sought to apply data augmentation approaches in deep learning, where transformed versions of the original data are generated to allow the learned deep neural network to be robust to data transformations [20, 18, 45, 17, 77]. One can also consider a set of possible transformations and explicitly transform the filters so that the feature maps are invariant to the selected types of transformations [78, 75, 50, 82]. Another family of approaches tries to generalize convolutional architectures by either extending the feature space to a group space of transformations [22, 15], or warping the input so that the transformation equivariance is implicitly encoded [32]. These approaches are limited since they 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.

Spatial transformer networks (SPN) [38] try to remove extraneous transformation variability a priori by introducing a spatial transformation module that is trained to recover the transformation of the input image and convert it to a canonical pose, independently of the underlying class, before feeding it to downstream layers. However, we argue that the transformation is not easily predicted directly from the image, and varies depending on the class label.

We present a novel extension of the deep learning framework to recover the optimal instantiations of the data while training for the downstream tasks. 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 SPN that produces a single transformed version of the original input, here we produce a transformed version for each class. The transformation is not predicted directly from the data, rather, for each class it is estimated to optimize the output of the unit representing that class.

We show that this framework can be easily applied to any existing neural network architecture and offers flexibility on the types of transformation considered by the model. We

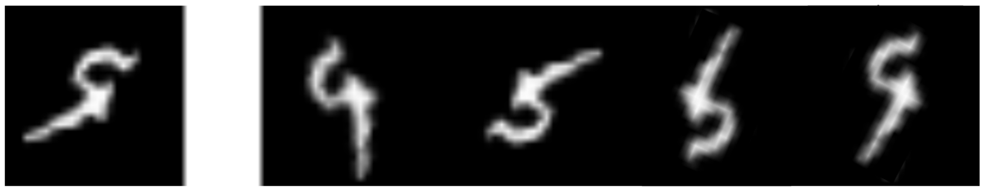


Figure 4.1: A rotated image of digit five can be further rotated to look like instantiations from digit class four, five, six and nine.

apply our framework to the training of convolutional neural networks, and present competitive results on mnist-rot, CIFAR-10, and the Google Earth dataset.

4.3 Background

In this section, we provide some background on spatial transformer network and discriminative latent variable models. In the sequel, we shall show the intuition of why the spatial transformer module cannot accurately estimate data transformations without capturing the data labels in some cases, and how we can overcome this problem by introducing latent variables to the model.

Spatial Transformer Network (SPN) [38] proposes a spatial transformer module that contains a localization network and a grid generator that can be trained end to end with a regular network. The localization network is trained to regress the transformation parameters for a given input. The grid generator creates the spatial grids based on the transformation parameters and warps the input feature maps accordingly. By combining the spatial transformer module with a regular convolution neural network, the whole network learns to produce spatial transformations for individual data examples during training for the downstream task.

However, sometimes the proper spatial transformation is undefined, making it difficult to estimate for the spatial transformer module. For example, in Figure 4.1 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 by analyzing the pose of the image object based on the class label. Therefore, we argue that an accurate estimate of transformation is not possible unless the image label is captured. In SPN, as the class labels of the images are not fully captured by the spatial transformer module (otherwise we do not need a downstream network to handle the classification task), the spatial transformer module cannot produce accurate spatial transformation for the input.

Discriminative Latent Variable Models: Introducing latent variables to discriminative models has been extensively studied in the framework of multiple instance learning (MIL), where the latent variables are used to capture the variations of many instances within a same labeled bag. The MI-SVM formulation of multiple instance learning was initially proposed in [2], and later reformulated as latent SVM in [21]. It considers a binary classifier that scores an example x as follows:

$$f_{\beta}(\mathbf{x}) = \max_{\mathbf{z}} \beta \cdot \Phi(\mathbf{x}, \mathbf{z}) \tag{4.1}$$

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. The label for $\mathbf{x}$ can be obtained by thresholding the score.

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.

4.4 Method

We consider a multi-class classifier that scores an example $\mathbf{x}$ for every class as follows:

$$f_{\beta_{\mathbf{j}}}(\mathbf{x}) = \max_{\mathbf{z}} \beta_{\mathbf{j}} \cdot \Phi_\theta(\mathcal{T}_{\mathbf{z}}(\mathbf{x})) \tag{4.2}$$

Here $\beta_{\mathbf{j}}$ is a vector of model parameters for class $\mathbf{j}$, Φ_θ is a feature mapping function parametrized by θ. The latent variable $\mathbf{z}$ is introduced to parametrize the transformations 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

$$\hat{c} = \arg\max_{\mathbf{j}} f_{\beta_{\mathbf{j}}}(\mathbf{x}). \tag{4.3}$$

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. As a result, we can optimize over the model parameters and 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:

$$\mathcal{L}(\Theta) = \sum_{\mathbf{i}} \max(0, 1 + \max_{\mathbf{j} \neq \mathbf{y_i}} f_{\beta_{\mathbf{j}}}(\mathbf{x_i}) - f_{\beta_{\mathbf{y_i}}}(\mathbf{x_i})) + \lambda(\|\theta\|^2 + \sum_{\mathbf{j}} \|\beta_{\mathbf{j}}\|^2) \tag{4.4}$$

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 instantiations of each example for separate classes. At first glance, it might seem that it would be simpler to forgo 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 4.1. Without competing with optimal instantiations of the data from non-target classes, the model might not be learning from the most competitive negative examples.

4.4.1 Optimizing the Training Loss

In order to minimize the hinge loss in Equation (4.4), 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 2.

When the latent variables form a discrete set, for example a finite set of rotations, optimization is performed by exhaustively search (ES). For continuous latent variables we optimize $f_{\beta_{\mathbf{j}}}(\mathbf{x}, \mathbf{z})$ from Equation(4.2) with respect to $\mathbf{z}$ by gradient descent (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 that optimal value over all initializations (ESGD).

A different approach, which is a direct generalization of the original spatial transformer model, is to use multiple spatial transformer modules (MSPN), one for each class, to directly predict optimal 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, in training this approach optimizes over the neural network parameters that predict the transformation. Then in testing there is no need for optimization as the optimal transformation for each class is predicted directly.

Algorithm 2 Two-Step Algorithm For Optimal Instantiation Learning

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

Unlike SPN 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, see Figure 4.2.

The methods GD,ESGD and MSPN 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.

4.4.2 Parametrization of Image Deformation

As we mentioned in the previous section, if we want to use the gradient to update the latent variables, we need to parametrize the image transformation in a form that is differentiable with respect to the latent variable $\mathbf{z}$. In this section, following the idea introduced in [1] and [38], we describe the general framework for the parametrization of image deformations.

Let us consider the image domain D as a continuum. An image can be defined on the domain D by $F(x^s)$, where $x^s \in D$. One can introduce a smooth deformation function ϕ, mapping from D to D. Therefore, a deformed image associated with the deformation function ϕ can be expressed as $\tilde{F}(x^t) = F(x^s) = F(\phi(x^t))$, where $x^t = \phi^{-1}(x^s) \in D$. In this case, for any point x^s in the original image F, ϕ^{-1} will deform it to the corresponding position x^t in the image $\tilde{F}$.

Now let us assume the deformation function ϕ is parametrized by $\mathbf{z}$. We can provide a kernel function defined on the domain of the prototype image to get a smooth version of $\tilde{F}(x^t)$. Specifically, we can approximate the value of $\tilde{F}(x^t)$ as follows:

$$\tilde{F}(x^t) = \int_{y \in D} F(y) K(\phi(x^t, \mathbf{z}), y)\, dy \tag{4.5}$$

Note that if $K(\phi(x^t, \mathbf{z}), y) = \delta(\phi(x^t, \mathbf{z}) - y)$ where $\delta(x)$ is the Dirac delta function, we have $\tilde{F}(x^t) = F(\phi(x_t)) = F(x_s)$. We can calculate the gradient of $\tilde{F}(x^t)$ with respect to x^s as follows:

$$\frac{\partial \tilde{F}(x^t)}{\partial x^s} = \frac{\partial \tilde{F}(\phi^{-1}(x^s, \mathbf{z}))}{\partial x^s} = \frac{\partial \int_{y \in D} F(y) K(x^s, y)\, dy}{\partial x^s} = \int_{y \in D} F(y) K'(x^s, y)\, dy \tag{4.6}$$

In SPN [38], the authors use the bilinear sampling kernel as K and denote D as the image

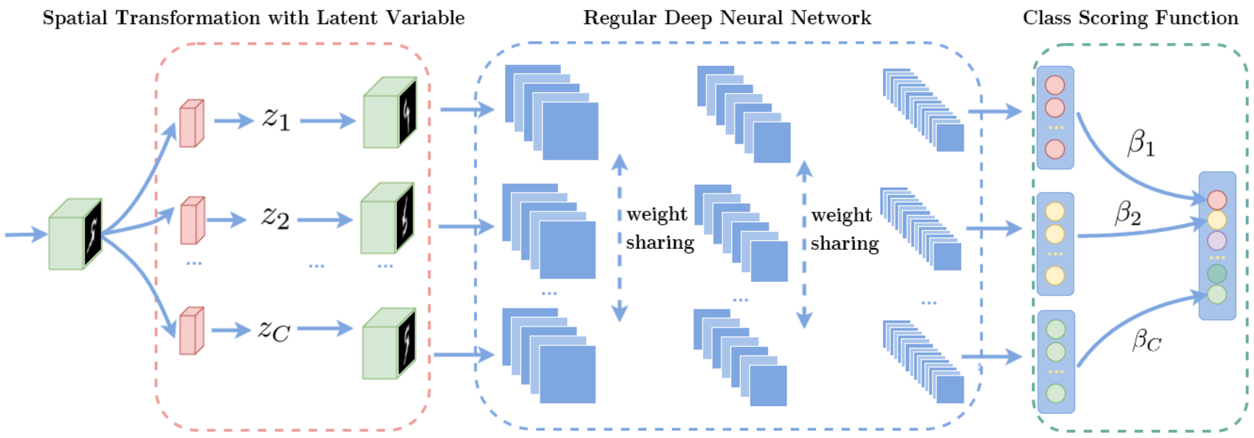


Figure 4.2: Model architecture for deep learning with optimal instantiations.

grid, then Equation(4.5) can be expressed as follows:

$$\tilde{F}(x^t) = \sum_{y \in D} F(y) \max(0, 1 - |x_0^s - y_0|) \max(0, 1 - |x_1^s - y_1|) \tag{4.7}$$

where (x_0^s, x_1^s) and (y_0, y_1) are the coordinates for x^s and y. The gradient with respect to the coordinate of the original image, $\frac{\partial \tilde{F}(x^t)}{\partial x^s}$, can also be easily derived from Equation(4.6).

To calculate $\frac{\partial \tilde{F}(x^t)}{\partial \mathbf{z}}$, the gradient with respect to $\mathbf{z}$, we can apply the chain rule $\frac{\partial \tilde{F}(x^t)}{\partial \mathbf{z}} = \frac{\partial \tilde{F}(x^t)}{\partial x^s} \frac{\partial x^s}{\partial \mathbf{z}}$. We need to explicitly define a deformation function ϕ that has $x^s = \phi(x^t, \mathbf{z})$, and then calculate $\frac{\partial x^s}{\partial \mathbf{z}}$ by $\frac{\partial \phi(x^t, \mathbf{z})}{\partial \mathbf{z}}$. In this paper, we use 2D affine transformation and thin plate spline transformation [9] as the two parametrizations for ϕ. The gradient of x^s with respect to $\mathbf{z}$ can be easily derived from the definitions of the transformation parametrizations.

4.4.3 Model Architecture

We put all the pieces together and show the model architecture in Figure 4.2. It includes three parts: a module for spatial transformation using latent variables, a regular deep neural network, and a module with class scoring functions. Given an input image $\mathbf{x}$, our framework first produces the optimal latent value $\mathbf{z_j}$ for each class $\mathbf{j}$, and creates the corresponding transformed image $\mathcal{T}_{\mathbf{z_j}}(\mathbf{x})$. The different transformed versions of the image are then passed to the weight-shared networks for feature extraction. Then the feature representation of the input $\Phi_\theta(\mathcal{T}_{\mathbf{z_j}}(\mathbf{x}))$ under each class $\mathbf{j}$ is passed to the last module to get the class scores. The label of the example can be determined by selecting the class that has the highest score.

Depending on which approach we use to obtain $\mathbf{z}$, the module for spatial transformation marked as the red region in Figure 4.2 involves a different structure and procedure. For MSPN the input will be passed to separate localization networks to predict latent values. For ES, GD and ESGD, a different procedure is used: the model transforms the input based on some initial values of $\mathbf{z}$, then passes the transformed input to the rest of the network to calculate the class scores or gradients so that the the latent values of $\mathbf{z}$ can be adjusted accordingly. This procedure will be repeated multiple times before the module finds the optimal latent values.

4.5 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.

4.5.1 The mnist-rot Dataset

The mnist-rot dataset is a variant of the MNIST dataset [46] 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. We initialize the weights of the CNN model by training it on a subset of the original MNIST dataset (first one hundred training image of each class). Then we train the CNN model with optimal instantiations on the mnist-rot training data. We experiment with three different approaches

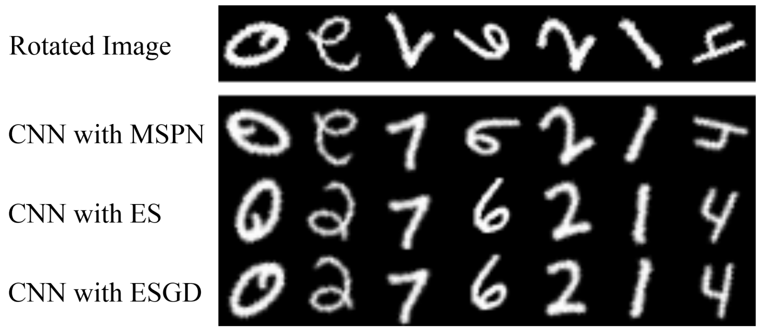


Figure 4.3: Correct the image rotations using the latent rotation angles estimated by three optimization approaches.

to optimizing over the latent variable including MSPN, ES and ESGD. In ESGD, we optimize the latent variable $\mathbf{z}$ in parallel to avoid getting stuck on local optimum, where $\mathbf{z}$ 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 4.3, 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 MSPN, the ES and 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 4.1, we show the error rate achieved by different models. When using class-specific spatial transformer modules to optimize over the latent variables (MSPN), we are able to achieve an error rate of 2.64%, significantly improved from 5.71% achieved by the conventional SPN. Our best result is achieved with ESGD, reaching an error rate of 1.25%. The state-of-art result 1.2% is achieved by TI-Pooling [45], where 24 explicitly rotated versions of the images are presented to the model for training and testing.

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

Model	Error (%)
TIRBM [72]	4.2
original CNN Model	4.1
Spatial Transformer Network (FCN)	5.71
TI-POOLING (24 rotations) [45]	**1.2**
CNN with MSPN	2.64
CNN with ES (8 rotations)	2.31
CNN with ESGD (8 initial rotations)	**1.25**

Table 4.1: The experiment results on the mnist-rot dataset.

experiment: We first train a traditional CNN model on 60000 training images of upright digits from the original MNIST dataset with multi-class hinge loss. Then the trained model can 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 CNN model trained with optimal instantiations. In Figure 4.4, we can 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 framwork effectively suppresses non-target class scores. While in the examples generated by the conventional CNN, we observe some undesired spikes of the scores for the non-target classes, which will lead to incorrect classifications. This approach achieves an error rate of 11.04%, which is far worse than any result we showed in Table 4.1.

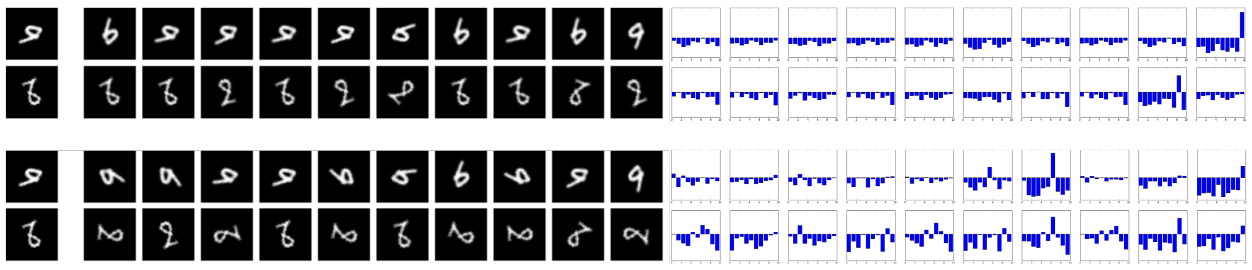


Figure 4.4: Examples of rotation-corrected images for ten separate classes using a conventional CNN trained on upright digits (bottom) and a CNN trained on rotated digits using our framework (top). For each rotation-corrected image, the corresponding class scores are shown on the right.

Model	Error (%)
CNN	3.17
SPN	4.9
CNN with GD (Thin Plate Spline)	**2.00**

Table 4.2: The experiment result on MNIST-100.

4.5.2 MNIST

We then train our model on the original MNIST dataset [47]. 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 MNIST1000*). In order to capture the local deformations of the data, we use the thin plate spline transformation as the latent variables. Similarly, we first initialize the CNN model by training it on 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.

In Figure 4.5, we show the optimal images transformed by the thin plate spline transformation for different classes, where some of the transformed images look like instantiations of other digit classes. As shown in Table 4.2, 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 SPN.

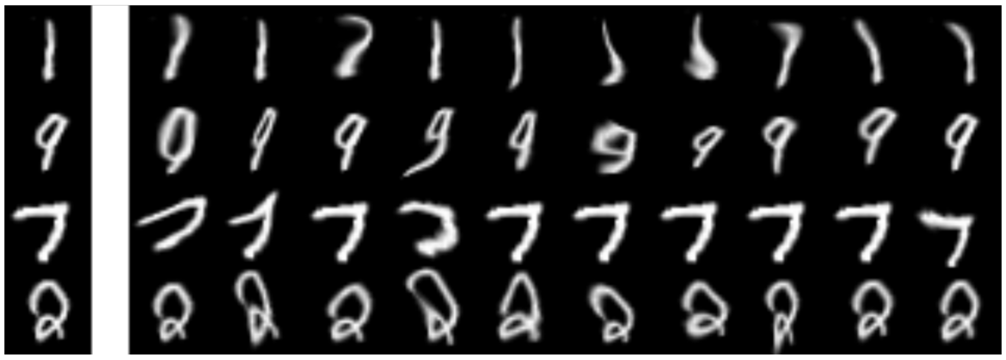


Figure 4.5: Examples of optimal images deformed by the thin plate spline transformation for different classes. The original images are shown in the first column.

4.5.3 CIFAR-10

Table 4.3: Experiment Result on CIFAR-10(400)

Model Description	Architecture	Error, %
DCGAN (semi-supervised approach) [61]		26.2($\pm$0.4)
Exemplar-CNN (semi-supervised approach) [19]	64c5-128c5-256c5-512f	24.6($\pm$0.2)
Exemplar-CNN (semi-supervised approach) [19]	92c5-256c5-512c5-1024f	23.4($\pm$0.2)
Steerable-CNN [16]	14 layers, 4.4M params	24.56
CNN Baseline	64c3-64c3-128c3-128c3-256c3-256f, 1.6M params	28.43
CNN with ESGD (Rotation)	64c3-64c3-128c3-128c3-256c3-256f, 1.6M params	27.9
CNN with GD (Translation, Scale)	64c3-64c3-128c3-128c3-256c3-256f, 1.6M params	25.53

We apply our model on the CIFAR-10 dataset [42]. 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. 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) for 2000 epochs and then train it using our framework for another 2000 epochs. We explore two experiments under this setting: 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 4.6, we show the optimal transformation via translation and scaling for the objects in the images from CIFAR-10. 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.

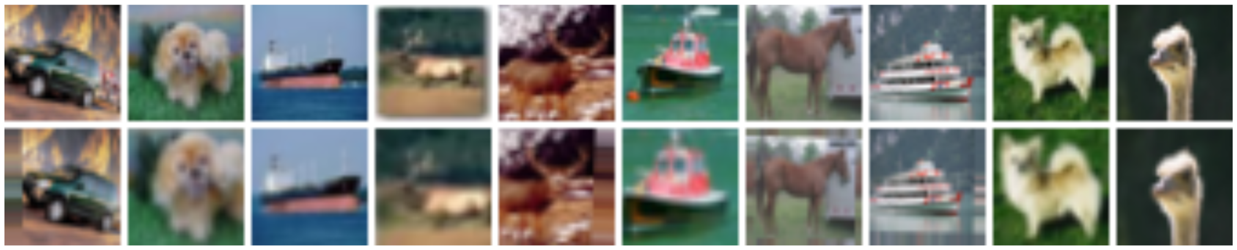


Figure 4.6: *Top:* Example images from the CIFAR-10 dataset; *Bottom:* Images translated and scaled by our model.

4.5.4 Google Earth Dataset

We then train our model on the Google Earth dataset [30], which contains aerial photos of streets with bounding boxes around the vehicles. Henriques et al. [31] also add angle annotation for each vehicle as a supplement of the dataset. The dataset contains 697 vehicles in 15 large images, where the first ten images are used for training and the rest for testing. The task of this dataset is to estimate the rotation parameter for each vehicle in the images.

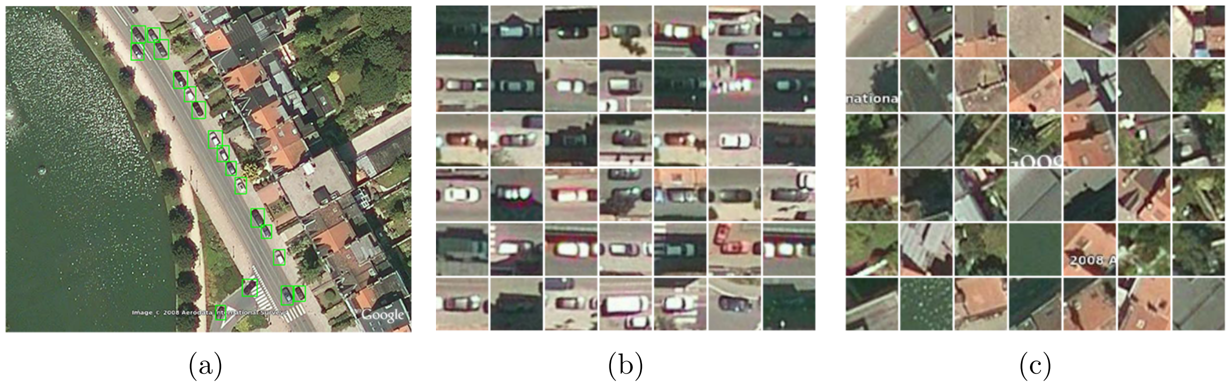


Figure 4.7: (a): An example of training images from the Google Earth dataset. (b) and (c) are examples of car images (car front point to the right) and background images we use for training a detection model for horizontal cars.

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 4.7, we show some image examples for training the detection model. We use this model as initialization to the ESGD training method, which further trains the model using images of rotated vehicles

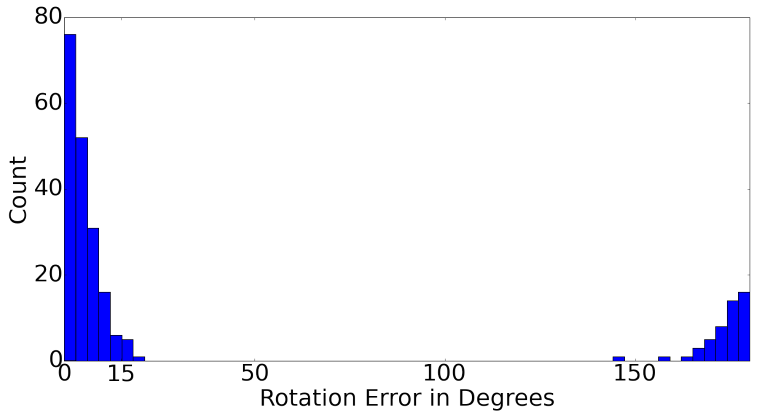


Figure 4.8: Histogram of rotation errors when estimating the rotation angles between $-180°$ to $180°$.

cropped from the training images. Then we can 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 [32], 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 4.4. We find that the CNN model with optimal instantiations outperforms the baseline model by a big 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 4.8, 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 regression result from [32] shown in Table 4.4 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. [32] define the rotation error as $e_i = \frac{\pi}{2} - \left| |\alpha_i \bmod \frac{\pi}{2} - \hat{\alpha}_i \bmod \frac{\pi}{2}| - \frac{\pi}{2} \right|$. We believe a better metric would be $e_i = \pi - \left| |\alpha_i - \hat{\alpha}_i| \bmod 2\pi - \pi \right|$ if $\alpha_i, \hat{\alpha}_i \in (-\pi, \pi)$, and $e_i = \frac{\pi}{2} - \left| |\alpha_i - \hat{\alpha}_i| \bmod \pi - \frac{\pi}{2} \right|$

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.

Model Description	Average rotation error (degree)
CNN for regression [32]	28.87
Warped-CNN [32]	26.44
CNN for regression ($-180°$ to $180°$)	63.7
CNN for regression ($-90°$ to $90°$)	43.1
CNN with ESGD ($-180°$ to $180°$)	37.8
CNN with ESGD ($-90°$ to $90°$)	4.87

Table 4.4: The average rotation errors of different models.

4.6 Recover the Support Masks for Objects

In this section, we describe how we can use deep learning with optimal instantiations to recover the support masks for the image objects. Such masks can be used for image decluttering.

4.6.1 Intuition and Method

As we described in the previous sections, latent variables can be introduced to the deep neural networks to effectively remove the nuisance transformations in the data. We parameterize the image deformation in a form that is differentiable with respect to the latent variables so that we can use the gradient to update the latent variables to obtain a reference pose for the object that is being classified.

As an example, in Figure 4.9, we show the comparison between the mean images of the original handwritten digits 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.

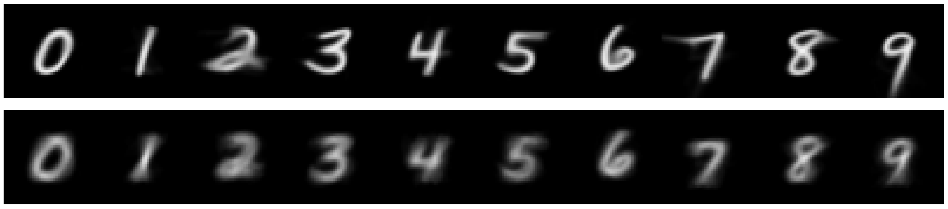


Figure 4.9: Mean images of handwritten digits (bottom) and pose-adjusted handwritten digits (top).

As we show in Figure 4.9, the reference poses recovered by this approach are well aligned and we can obtain the support map of each object class by taking the mean images of the pose-adjusted objects. In the following, we will show how we can apply the support maps to remove clutter from images.

4.6.2 Robust to Clutter

In this section, we want to investigate if our approach is robust to different types of clutters, which are not observed in the training data. Assume we train a CNN model with optimal instantiations using the MNIST training data, we explore if this model can successfully recover the reference poses for the test image objects when one of the following two type of clutters exists in the images:

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 detection.

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

We show some examples of images with the two different clutter types in Figure 4.10. Note that nearby clutter will not touch or overlap with the original digit in the center.

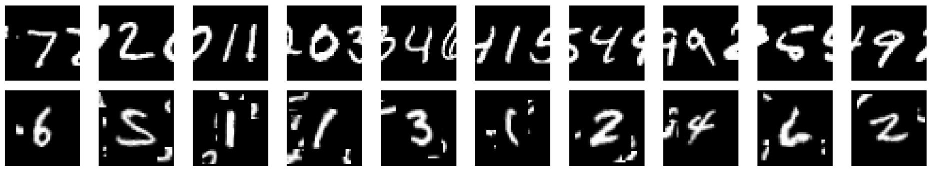


Figure 4.10: Sample images of two different types of clutters: flanking digits (top) and random clutter (bottom).

We first show the reference poses recovered by our approach for images with flanking digit clutter in the first two rows of Figure 4.11. Only the reference poses of the images under the correct class labels are shown here. Although the model is not able to completely remove the clutter around the target digits, the model adjusts 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. Therefore, the size of the pose-adjusted test digits should remain the same, and our model is not able to remove the nearby clutter by zooming in on the center digits in the images. If the model is trained on images with rescaled objects that occupy the entire canvas without any padding, then the model is able to zoom in on the center digits and remove the surrounding clutter in the test images.

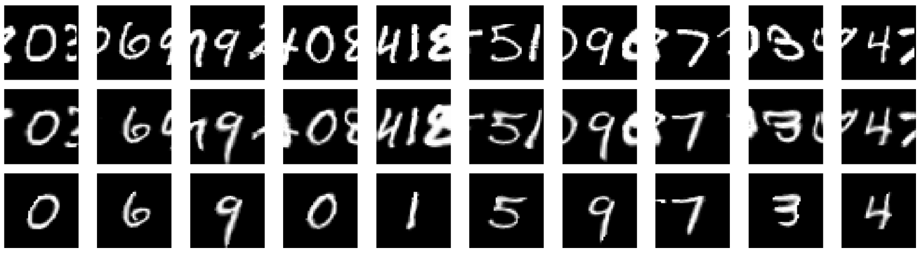


Figure 4.11: Examples of the original images with flanking digit clutter are shown in the first row. The corresponding recovered reference poses under the correct class labels are shown in the second row. The decluttered images extracted by applying object class support are shown in the third row using a decluttering approach described in Section 4.6.3.

We then show the reference poses recovered by our approach for images with random clutter surrounding the target objects in the first two rows of Figure 4.12. Similarly, the model can adjust the pose of the target object in the center regardless of the surrounding random clutter that the model does not observe in the training data. It is worth noting that the reference poses captured for the objects with surrounding random clutter are different from those captured for the objects with flanking digits. As we will discuss in the following section, we will show some evidence to prove that our approach is less robust to random clutter and the reference poses captured here are not perfect.

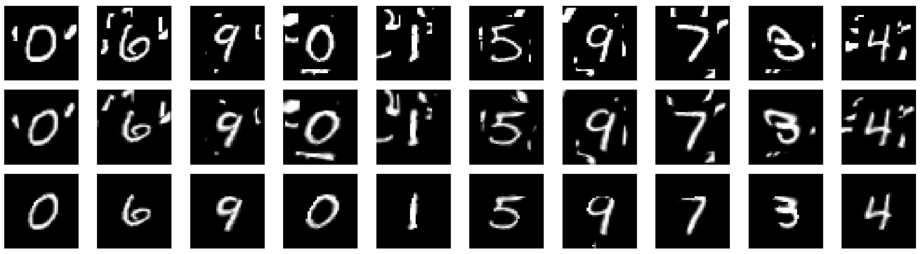


Figure 4.12: Examples of the original images with random clutter are shown in the first row. The corresponding recovered reference poses under the correct class labels are shown in the second row. The decluttered images extracted by applying object class support of the correct class are shown in the third row using a decluttering approach described in Section 4.6.3.

4.6.3 Declutter Images with Object Class Support Map

As we showed in the last section, although the reference poses are obtained using our approach, the surrounding clutter still exists as our approach is not able to zoom in on the center digits in the images and remove the clutter. In this section, we show that we can obtain decluttered images using the support maps of the corresponding object classes.

When the reference poses of the image objects within the same class are well aligned, as we show in Figure 4.9, the mean images of pose-adjusted objects can be used to calculate the support map of the reference pose for each class. As the mean images of pose-adjusted objects are much sharper, we are able to obtain support maps that are more precise. If the object labels of the images are known, we can apply the support maps of the correct classes to the pose-aligned images with clutter and obtain the decluttered images, such as shown in Figure 4.11 and Figure 4.12. Note that this decluttering step is naturally achieved with the pre-trained model without any supervision. 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 4.12, 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 with the support masks of the corresponding class, indicating that the surrounding clutter makes our approach less stable.

4.6.4 Classify the Decluttered Images

We now show how decluttering images with object class support masks can be incorporated into our framework to solve classification tasks for images with clutter. Intuitively, it would be ideal if we can remove the clutter in the images before we classify them. As we showed above, we are able to obtain the support maps for each object class and use the support maps to remove the clutter in the images if the labels of the images are known. However, how can we apply the support maps to declutter a test image when the label of the target object is unknown?

Recall that in our approach, we optimize over the latent variables to recover the reference pose for each object, and the latent variables are optimized for each class separately. As a result, we know which object class the model is optimizing for during the optimization process. Therefore, for each object, a reference pose under a particular class can be obtained by our model, and we can then apply the support map of the corresponding class to the reference pose of the object 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

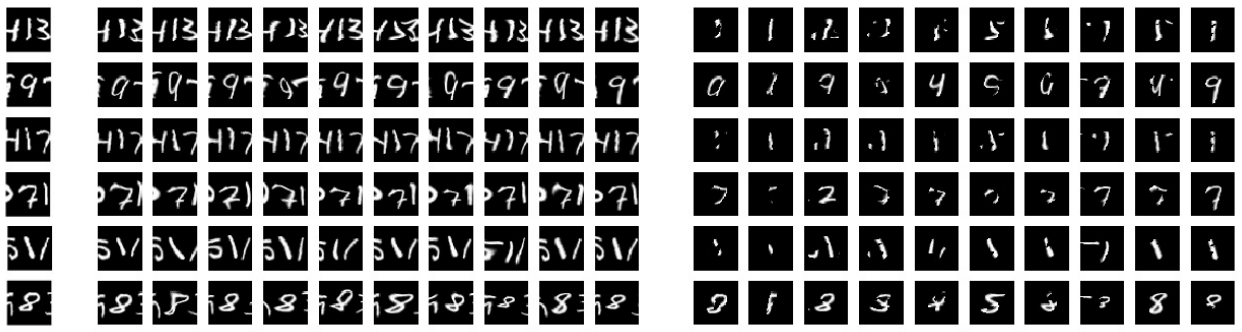


Figure 4.13: We show examples of misclassified digits and the corresponding optimal images captured for the corresponding images for different classes (in the middle). On the right, we show the corresponding decluttered images that we feed to the downstream network to produce class scores for classification.

trained on the MNIST1000 dataset with a clear background and tested on original test dataset with flanking digit clutter, our approach improves the classification accuracy rate from 89.82% to 91.07% when we remove the clutter from test images using the support maps. However, if we test the model on the original test dataset with random surrounding clutter, the classification accuracy rate drops from 88.59% to 86.91%. This again shows that our approach is less robust to random surrounding clutter.

We then look into the mistake cases made by our algorithm. In Figure 4.13, we show examples of misclassified digits using the classification approach we described above. As we can see from the figure, there are two types of mistakes: First, the subset problem: for some object classes, the shapes could be similar to some image parts contained in another object class. For instance, as we can see from the examples in the second row of Figure 4.13, a digit nine could look like digit zero, digit four and digit seven 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. Second, some mistakes are caused by some undesired deformations. For instance, in the examples we show in the first row of

Figure 4.13, we observe in the six column that clutter from the nearby region gets pulled to the digit one in the center to form a new object that looks like a digit five. After we apply the support map to the image and remove the clutter, this image looks exactly like a digit five. This is caused by too much flexibility of the deformation allowed in the thin-plate spline methods, which we can alleviate by regularizing on the degree of deformation allowed by the thin-plate spline.

4.6.5 Resolving the Subset Problem

We further seek methods to resolve the subset problem. As we discussed 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. Therefore, this architecture is not able to resolve the subset problem that we experienced above.

To resolve this, we apply a two-step mechanism for training and testing the images. The first step of our method is the same as our learning mechanism with optimal instantiations, where the optimal instantiations of each example for separate classes are captured, and we can apply the support maps of different object classes to obtain the optimally deformed images for separate classes with the clutter removed. In the second stage, for each example, we stack the images of its optimal instantiations for the separate classes and train a regular CNN to classify the label of the example based on the stack of input images - one for each class. This time, since the model is able to observe the optimal deformed and decluttered images from all the classes, it has much richer information on what get masked out by the support maps, and it could 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, which are 2.3% and 4.95% higher than the approach without the two-step mechanism.

4.7 Deep learning with Optimal Instantiations for 3D Data

With the recent development of 3D scanning, extra-sensor camera (Kinect, LiDAR laser scanner, etc.) and other techniques, people are able to collect more 3D Data. Such data can be useful for vision tasks in the 3D environment, with applications in self-driving cars, robotics, augmented reality, etc.

One interesting topic is to detect the poses of 3D objects. For example, by knowing the precise pose of a 3D chair (location, facing), a robot can come up with motions to sit on the chair. Most research formulates this problem as predicting the object's class and the 3D pose based on a 2D or 3D input image (2D image with depth or 3D data in other data types, described in Section 4.7.1). Several types of approaches have been proposed to solve this problem. The most traditional approaches are template-based approaches where a template is scanned across the space and a distance measure is designed to find the best match with the object and the pose[34, 35]. One can also detect the interest points of objects, describe them with local features, and then match them with the templates in the database to capture the object identity and pose [37, 8, 10]. However, most of these approaches assume that the 3D shape of the object instance is known, and they infer the 3D orientation by matching the shape of the 3D template and the data. OctNet[64] relaxes this setup by assuming only the object category is known, and learns to regress the orientations using CNNs. However, it is not able to accurately estimate the 3D orientations when the object category is unknown. This is expected if the model is trained to regress the 3D orientations without knowing the object class labels since the model might be confused about the orientations of objects from different classes and the inter-class variance is more difficult to capture in the 3D data.

In this section, we describe how to predict the object classes and the 3D poses when the volumetric representations of the 3D data are given. Similar to what we have observed in the 2D cases, we show that we can recover the spatial transformation of the 3D objects while capturing their class labels by applying deep learning with optimal instantiations.

4.7.1 3D Data Description

There are several data types representing 3D objects, and different approaches of 3D object detection and pose estimation have been designed for different types of 3D data representation. We list a few of them below:

RGB-D data: RGB-D sensors are capable of providing depth information for each pixel. Although the information provided by such a data type is not as rich as real 3D data where we should have the information of all the observable surfaces, it still provides more information than traditional 2D images and could lead to better performance. Lots of RGB-D datasets, as well as corresponding detection algorithms, have been proposed, including [44, 33, 7, 25].

Point clouds: Usually collected by 3D scanners such as the LiDAR laser scanner, point cloud data provides richer spatial information for the objects compared to the RGB-D data. The 3D scanner measures a large number of points on object surfaces in a three-dimensional coordinate system and outputs a point cloud as the data. Since the data is represented by an unordered set of points, the data representation is invariant to any permutations within the set, and it might lead to some problems in real applications. Therefore, point-cloud data is usually converted to a voxel-based representation for further processing. Recently, a few methods are proposed to resolve this by introducing new data structures to store the point cloud data and have achieved good performance [62, 41].

Descriptions of composing polygons: This is usually used in computer graphics to describe a 3D model. A 3D model states the coordinates of the vertices, the groups of vertices that form the polygons and coloring of the polygons. Although this is sufficient to describe any 3D data, in general one would convert it to other data types that are easier to interpret and represent for data modeling.

2D views of 3D objects: One can also create 2D views of 3D objects to describe the 3D objects. For example, the panoramic view is a cylinder projection of 3D objects to 2D images around their central axes. Although such a 3D data representation is not invariant to 3D rotations, it captures the shape and the texture information of the 3D object and therefore can be used for 3D object recognition tasks [57, 70, 69]. Similar to the panoramic view of 3D objects, one can adjust the camera position to capture multiple views of 3D objects to obtain the information of the objects using 2D images. A few camera positions are picked, and the views of the 3D objects are captured to represent the shape and texture information of the 3D objects. Similarly, multiple 2D views of the 3D objects can be combined to represent the 3D objects [73, 39, 60]. Once the 2D views of the 3D objects are obtained, one can use approaches that are commonly used for the 2D data to resolve the vision tasks for 3D data.

Volumetric representation: The most common way to represent the 3D objects in the literature is the volumetric representation, which can represent a 3D object on a 3D voxel grid. Although the volumetric representation is not able to capture the texture information of the 3D objects and requires voxel grids that have high resolution to exploit the detailed shapes of the 3D objects, it has gained popularity as the volumetric representations of the 3D objects can be stored and manipulated with simple data structures. Lots of algorithms have been proposed to resolve the vision tasks for 3D data using volumetric representation [79, 52, 64, 68, 11, 60].

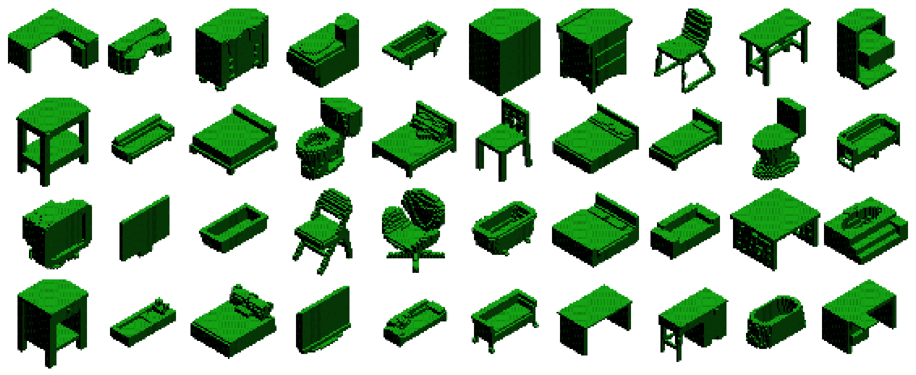


Figure 4.14: Example images of volumetric representation of 3D objects from the ModelNet dataset.

In this work, we use the volumetric representation of the 3D object as the data for our model. In Figure 4.14, we show some examples of volumetric representations of 3D objects.

4.7.2 Method

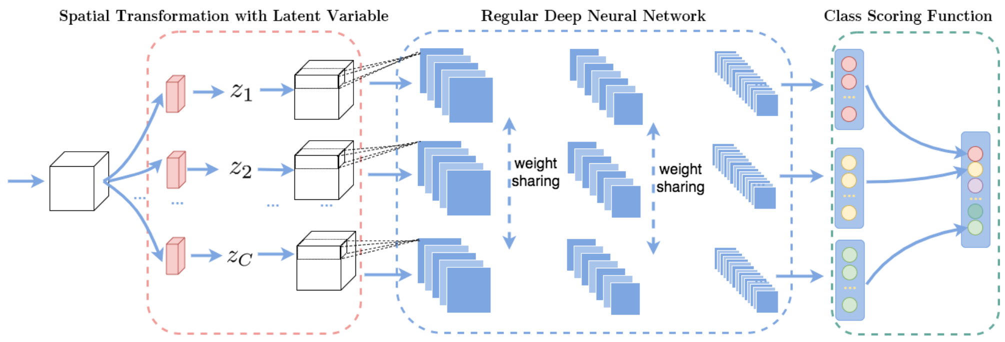
Similar to the scenario in 2D, we consider a 3D object classification problem where the input data is the volumetric representation of 3D objects. We design a convolutional neural network architecture to solve the classification task.

For 2D images, it is common to design a CNN architecture where the filters in each convolutional layer perform 2D convolutions for the incoming feature maps. It is natural to extend this approach to the 3D data. We pick one dimension of the volumetric representation of the 3D object as the "depth" channel (like color channel in 2D images) and feed the 3D data into a CNN architecture as if they were 2D images.

To understand why such a design makes sense, let us consider a CNN architecture where the first convolutional layer contains filters of size 1×1. Intuitively, it acts as a scanner to perform 3D to 2D projection along the "depth" channel. Therefore, such an architecture can capture the exterior appearance of an object. This is usually sufficient to classify an object as we normally do not depend on the interior structure of a 3D object to determine its category (at least not in this work).

Similar to our previous work, we incorporate latent transformation variables to the convolutional neural network to enable learning with optimal instantiations. We show the model architecture in Figure 4.15. As in Section 4.4, a two-step algorithm is required for optimal instantiation learning. During training, this framework takes the 3D object data as input and alternates between optimizing over the latent variables to find the optimal instantiations for each class and optimizing over the model parameters to obtain a better classification model. During testing, the model passes the input through the network to calculate gradients so that the latent values can be adjusted accordingly. Eventually, the optimal latent value z_j under each class j together with the corresponding class score of the pose-adjusted 3D object are captured by the model. The label of the example can be determined by selecting the class that has the highest score.

We pick a dimension as the "depth" channel and treat the 3D object data as 2D image data with a long "depth" channel. Therefore, we can again apply the architecture of 2D convolutional neural network to the 3D object data.


Spatial Transformation with Latent Variable
Regular Deep Neural Network
Class Scoring Function
z_1
z_2
z_C
weight sharing
weight sharing
Figure 4.15: The CNN architecture for 3D object data classification.

Note that the module for spatial transformation marked as the red region in Figure 4.15 requires a transformation parameterization that is differentiable. In the next section, we will discuss the parameterization of 3D spatial transformation we use in this work.

4.7.3 Parameterization of 3D Spatial Transformation

In the 2D scenario, we explored the transformation parameterization using an affine transformation matrix and thin plate splines. Such parameterization methods can also be applied to the 3D scenario. For example, we can parameterize a 3D geometric transformation using a transformation matrix as follows:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

It requires 12 latent variables to capture the geometric transformation of a 3D object. In our work, we implement a 3D spatial transformer layer that uses only three latent variables to capture the degrees of rotation along each of the three axes. In Figure 4.16, we show how a voxelized 3D object can be randomly rotated in three axes.

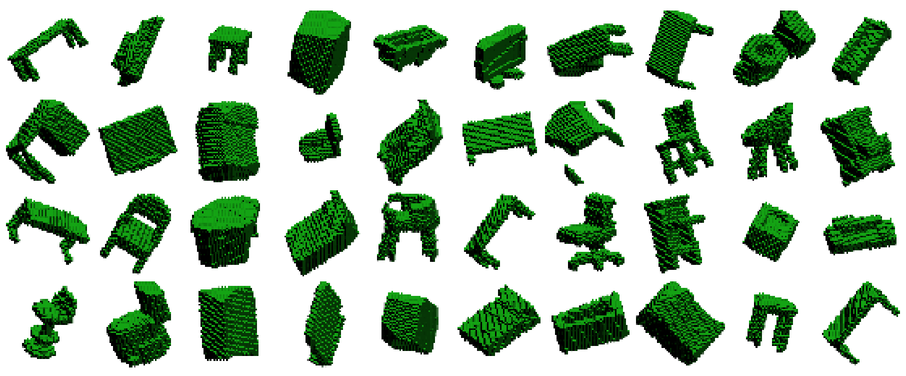


Figure 4.16: Randomly rotated 3d objects.

4.7.4 Recover the Reference Poses of 3D Objects

In this section, we explore how our approach can be used to recover the reference poses of 3D objects while classifying the objects. In this work, we use the ModelNet-10 dataset [79] that contains orientation-aligned 3D CAD models that belong to 10 different categories. Some examples of volumetric representations of the objects in the dataset can be found in Figure 4.14. We randomly rotate the 3D objects in the testing data along each of the three axes by degrees between -45° to 45° and we want to see if our approach can recover the correct reference poses of the objects.

We choose the angle of rotation along each axis as the latent variable. We first initialize our model by training on the orientation-aligned training data from the ModelNet-10 dataset and then train our framework with randomly rotated training data. We optimize the latent variable z in parallel to avoid getting stuck in local optimum, where z is initialized for each axis at three different rotations $(-30^\circ, 0^\circ, 30^\circ)$, and each is optimized for ten iterations using gradient descent. The value of z that produces the highest score is chosen as the optimal value. During testing, similarly, we optimize the latent variable z to produce the highest score for separate classes.

We show the pose-adjusted objects for different classes in Figure 4.17. The first column shows some rotated 3D objects, which are randomly rotated from the orientation-aligned objects that are shown in the second column. Starting from the third column, we show the recovered poses for different classes and the red square frames indicate the pose-adjust objects for the target classes. It is clear that our approach is able to recover the poses that are close to the original orientation-aligned objects.

4.8 Conclusion

In this work, we propose a framework for training deep neural networks with optimal instantiations of the data. By introducing latent variables to parametrize the transformation of

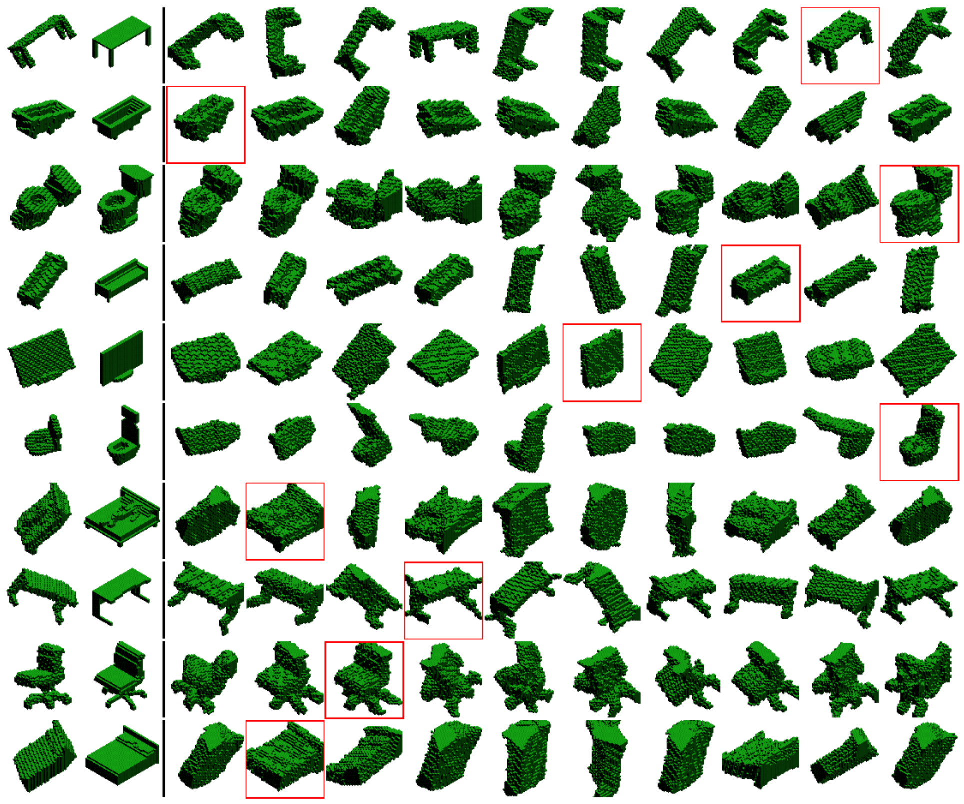


Figure 4.17: Recovered reference pose of 3d objects.

the data for each class, our approach is able to better obtain the reference pose for the object that is being classified. We show such an approach can be easily applied together with any existing neural network architecture and is compatible with general types of transformations including rotation, translation, scaling and local deformations.

CHAPTER 5

LEARNING FROM 3D CAD MODELS

5.1 Introduction

In the previous chapters, we discussed how to make architectural changes to the existing deep CNN models to achieve transformation invariance. Although lots of improvements have been made to allow traditional deep CNN models to capture the transformation variations that are not observed in the training data, these approaches are still heavily dependent on large-scale training datasets. Since labeled data is difficult to collect, the performance of deep CNN models could be limited due to the lack of data.

In this chapter, we want to discuss the possibility of using 3D computer-aided design (CAD) models to render 2D training images as a data augmentation approach to help train deep CNN models. Thanks to the development of modern 3D graphics programming, the 3D CAD models become easier to acquire, and they can be used to generate synthetic 2d images. By generating 2D images from 3D CAD models, we can obtain a more compact way of representing an object class by rendering images using different postures, camera positions, material textures and light positions. This allows more transformation variations in data to be observed when training the models.

Although the synthetic images can capture the shapes of the objects, they usually lack realism compared to real images. Objects in synthetic images are usually perfectly isolated, and the texture, poses and lighting are less realistic. Therefore, models trained on such data usually fail to deliver competitive results. In this work, we explore the performance gap between the models trained on real images and the models trained on synthetic images, and we also propose several methods to bridge the gap. We investigate a method for zero-shot or few-shot learning of the image objects which requires less human-labeled data, and we explore how much we can improve compared to training with real data.

Another effect on the realism of the synthetic images comes from the compatibility between the rendered object images from 3D CAD models and the background images that are usually selected from some existing data source. On the one hand, it is not feasible to manually select background images for all the generated object images since our goal is to automate the creation of the large-scale synthetic image dataset. On the other hand, if the background images are randomly selected, they might contain undesired objects or noise in the image scenes, leading to unrealistic images when combining them with the rendered objects. In order to resolve this problem and reduce the bias and noise introduced by the background of the images, we also create a model that can learn to produce support masks and extract the target objects from the images. By doing this, a classifier trained on the synthetic images of rendered objects without background can be applied to classifying the extracted target objects from the real images. We will discuss how this can be applied to real image classification.

The rest is organized as follows: In Section 5.2, we will discuss some recent work on generating synthetic images to help improve the performance on vision tasks. We will also discuss some related work on pixel-wise prediction that can be applied to produce the support masks for object extraction. After that, we will describe two approaches by using 3D CAD models to help 2D image classification tasks, including augmenting the training data with synthetic images and learning to find the support masks for the target objects in the images. The rationale behind these two approaches and the corresponding methods we use will be discussed in Section 5.3 and Section 5.4. The details on rendering synthetic images from 3D CAD models are discussed in Section 5.5, and we present the experimental results and conclusions in Section 5.6 and Section 5.7 respectively.

5.2 Related Work

There is plenty of literature on how to create synthetic images from 3D CAD models, exploring different ways of taking advantage of the enormous amount of generated images to help resolve computer vision tasks. Here we describe the details of some of these methods and discuss the difference between our approach and the approaches in the literature. In addition, as we mentioned above, we also seek methods to extract target objects from the real images so that we can adopt models that are trained on synthetic images without background. This is related to the image segmentation problem, and we will discuss some possible solutions for that in the literature.

5.2.1 Exploring the Advantage of Using 3D CAD Models

Thanks to the flexibility of generating synthetic images, one can use it to analyze the transformation invariances of the learned models. In [59], in order to show whether the transformation invariances can be learned by deep neural networks for object detection, the authors generate synthetic images under different appearance factors including rotation (azimuth, elevation), size, occlusion, and truncation. The paper shows that the overall performance of deep neural networks improves when additional synthetic images are provided as training data, indicating the advantage of using synthetic images. It shows, however, that the model underperforms for the truncated, occluded and small objects even when additional synthetic images are provided, suggesting that structural changes to traditional deep neural networks are needed for these specific types of data variations.

In a similar vein, Peng et al. [58] analyze the invariance to cues (including object texture, color, pose and shape) in deep neural networks with the help of 3D CAD models. To see whether the deep neural network is able to extract the same high-level features despite different (or missing) cues, the authors design a comparison experiment, where two training datasets of synthetic images are created, one with a particular cue and the other without. A trained deep neural network is used to extract the high-level features of images from these two datasets, and the extracted feature maps from both sets are used to train two object detectors. A similar detection performance between the two detectors would indicate that the deep neural networks are invariant to this particular cue whereas a less cue-invariant network will lead to poorer performance. The paper shows that the deep neural networks are less invariant to these cues when the network is not fine-tuned for a particular task. Even when the network is fine-tuned, the network shows a significant boost from adding more variations of shapes, postures and object textures, indicating the model is not entirely invariant to these factors. By augmenting the real data with synthetic images, the authors also show a significant performance improvement by using 3D CAD models for detection tasks in the scenario when zero or few real training examples are available to train the detectors.

Similarly in [81] and [51], the authors train pedestrian detection models using images from virtual scenarios [1] and show that the learned detection models can be successfully applied to pedestrian detection in real images. In a separate line of research, much work has been using 3D models to render images for the viewpoint estimation task. In [3], the authors create a large-scale dataset of view-dependent 2D synthetic images from 3D object models. Given a 2D image, the authors design an approach to find an alignment between the 2D image and the most similar 3D model rendered at the most appropriate camera viewpoint. In [74], the authors design a deep CNN for the viewpoint estimation task and show that training CNN with a large amount of synthetic data is effective for such a task.

In this work, we follow the ideas of augmenting real training images with synthetic images and investigate how synthetic images could help the classification task when zero or few real images are available during the training. As we mentioned above, such an approach is not perfectly applicable since the real photos contain complex environments as background and

1. See Section 5.2.2 for the details of generating synthetic images by this method.

they are not easily available in synthetic images. Therefore, we explore the approach of foreground object extraction to resolve this problem.

5.2.2 Synthetic Image Generation

The first step in the work we described above is to generate synthetic images from 3D CAD models. In [59], the authors use the PASCAL3D+ dataset[80] for empirical analysis, which contains aligned 3D CAD model annotations for the 2D images in the PASCAL VOC 2012 dataset. Three different rendering types of synthetic images are investigated by the authors, including the wire-frame images, the plain-texture images, and the texture-transferred images. Each of these types represents a different level of richness of image detail: Wire-frame images only provide the shape boundaries of the image objects, which only reflect the shapes of the target objects without showing the textures and backgrounds. Plain-texture images are generated by rendering programs, where a plain color is applied to the objects so that the synthetic images do not have realistic textures. Texture-transferred images are generated with a distribution that is more similar to the real data: Since each image is annotated with the target object and the corresponding aligned 3D CAD model, one can replace the existing object with a new 3D CAD model of the same object class and apply a different texture to it. Therefore, the objects in the generated images will have different shapes and textures, and the synthetic images are much more realistic.

Similarly, as we mentioned previously, [58] tries to investigate if the deep neural network is invariant to different image textures and various background. To study this, the authors generate the synthetic image under different settings, where the target objects can be rendered with real textures or gray textures, combined with different types of backgrounds (images with real background, white background, and gray background). 3D CAD models from the desired object categories are selected and downloaded from the 3D Warehouse[2],

2. The SketchUp 3D Warehouse is an open source library where people can share 3D models online

and the viewpoints of the 3D models are manually selected before rendering the 2D images. In order to create realistic object textures for the images, the authors extract the textures of the same objects from real images and stretch them to fit the CAD models. To generate realistic background scenes, they gather real images of scenes where each object category is likely to appear, and the selected images are combined with the corresponding rendered image objects to generate the final synthetic images.

Other approaches are used to generate synthetic images as well. In [81] and [51], the authors create virtual scenarios from video sequences generated by a video game engine, including realistic scenes with roads, buildings, vehicles, pedestrians, etc., under different illumination conditions. Such generated synthetic images are more realistic as they are captured from the video sequences of some video games. However, it is usually not easy to add 3D models of desired objects to an existing video game engine, making this approach less flexible.

5.2.3 Image Foreground-Background Segmentation

As we mentioned earlier, in order to apply a model trained with synthetic images of rendered objects without background to classifying real images, we need to extract the target objects from the real images. This problem is equivalent to the semantic segmentation problem if we only need to label the scene with two categories: the image foreground that contains the object and the image background.

Over the past few years, much work has been done on the semantic segmentation problem using deep learning approaches. For example, R-CNN[24] generates region proposals by selective search [76], and each region is warped to the desired resolution before being sent to a pre-trained CNN for feature extraction. The computed feature map for each region can then be used for fine-tuning the CNN and learning object detectors. Bounding-box regressors are learned to improve the localization performance further. During testing, selective search on the test image is again used to extract region proposals and each proposal is warped and fed to the CNN to compute class scores. It is then followed by a non-maximum suppression procedure that is adopted to reject overlapping regions. This is extended in [23] by replacing the multi-stage training in R-CNN with a single-stage training procedure using a multi-task loss that can guide the training of the bounding box detection and the object classification. Later in [63], the R-CNN architecture is further unified with a region proposal network to compute region proposals instead of generating them by selective search. Note that these approaches do not directly provide semantic segmentation with arbitrary shapes but rely on other region proposals such as CPMC [12] to create segmentated regions with arbitrary shapes. The bounding boxes surrounding these regions are then passed to the network to achieve the labels of these regions. A recent update of this line of work, Mask R-CNN [29], allows the network to produce a pixel-wise binary prediction along with the object classification and the bounding box regression, which decouples the mask prediction and the class prediction, and it prevents competition among different classes when generating semantic segmentation. Similarly, in order to produce semantic segmentation while detecting the objects, [27] employs a bottom-up image segmentation by first generating the region proposals followed by a region refinement step to create semantic segmentation at a fine resolution.

Another line of research uses CNN features to provide dense category-level pixel labels. For example, [49] replaces the fully-connected layers with the convolutional layers in the CNN and builds a fully convolutional network, and the network will produce a heat map as the output for the pixel-level prediction. However, due to the pooling and down-sampling operations in the CNN, the resulting heat map usually has a different resolution compared to the input. Therefore, the authors propose deconvolutional layers that can be used to upsample the output heatmap to match the size of the input. In addition to that, skip connections are introduced so that the resulting segmentation maps will have a high resolution.

Similarly, [28] proposes to use skip-layers, where the output feature maps of the selected layers are upsampled to the same resolution before being concatenated together to form a hypercolumn descriptor for each pixel. The hypercolumn descriptor can later be used to train a pixel-wise classifier to determine the pixel category. An encoder-decoder architecture can also be adopted [5, 56, 66] to produce the pixel-wise prediction for the semantic segmentation task.

In this work, we follow the idea in [28] and use the hypercolumn descriptors from the CNN to produce the pixel-wise prediction for foreground-background segmentation. We will discuss more details of this approach in Section 5.4.

5.3 Few-shot Learning with Synthetic Images

As we discussed earlier, compared to the limited amount of hand-collected real images, the richness and compactness of representing an object class using synthetic images are appealing. One can easily collect 3D CAD models of the desired categories and create views of these models using rendering programs. Therefore, augmenting the training data with synthetic images generated from 3D CAD models could allow the model to observe more transformation variations from the synthetic data that are not readily available from the real data. In real images, target objects usually appear on particular scenes, and it is crucial for the model to be robust to various backgrounds while recognizing the target objects in the images. Note that the rendering program is only able to render the desired object in front of an empty background unless we have 3D CAD models for the background. Therefore, in order to make the rendered images more realistic, one also needs to generate realistic background images in addition to the views of the target objects. We will discuss in detail on how to generate synthetic images in Section 5.5.

To show how this could help the 2D image classification tasks, we can train a classifier using synthetic images and see if the trained classifier could be used for classifying the real images. We introduce a scenario when limited amount of real images are available for the training (few-shot learning), and we investigate if augmenting the real training images with the synthetic images could help improve the result.

We train a convolutional neural network as the classification model. In Figure 5.1, we show the model architecture of the convolutional neural network for the input images of dimension $32 \times 32 \times 3$. The trained model will be further used to classify the real images.

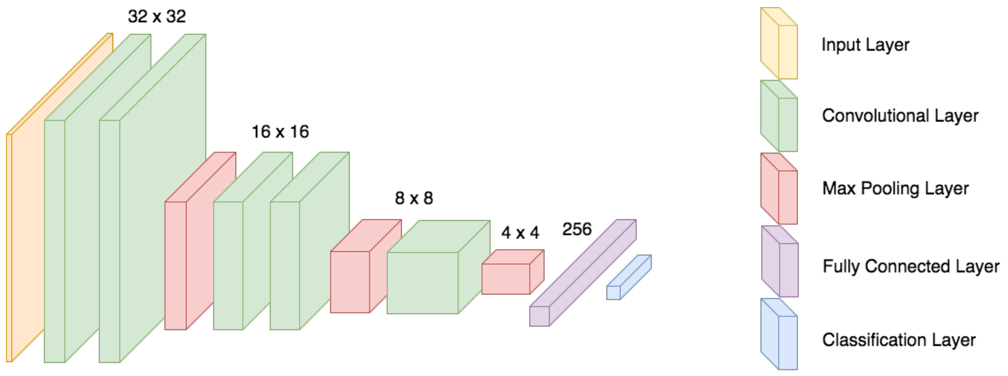


Figure 5.1: Model architecture of the convolutional neural network we use for classifying images of size 32×32.

5.4 Learning Image Object Extraction

5.4.1 Intuition

As we mentioned in the previous section, one of the difficulties in creating synthetic images is that we also need to generate realistic background images that can not be rendered as they are not included in the 3D CAD models. Indeed, we can easily collect unlabeled real images as background images and combine them with the rendered object images to create synthetic images, yet such an approach is still not able to create very realistic images. It would be preferable to have a model trained with synthetic images of objects without background directly apply to classifying real images that have noisy background.

One way to achieve this is to design an algorithm to segment the real images into the foreground regions (target objects) and the background regions and then extract the target objects from the test images. Once the background regions of the test images are removed, we can directly apply a model trained with synthetic images with clear background to classify the target objects extracted from the real images. We want to emphasize that the segmentation model also needs to be trained on the synthetic images, as we want to keep the supervision from the real training images at a minimum level.

Such a foreground-background segmentation can be achieved by learning a pixel-to-pixel prediction model, where a binary prediction is generated for each pixel to form the support mask. We also want to emphasize that the model is trained to perform the segmentation for all the objects despite the labels of the objects. We will discuss how we can build a learning-based system that generates the support masks for the target objects in this section.

5.4.2 Learning the Support Mask via Pixel Classification

We frame the support mask learning problem as learning a function $f : \mathcal{X} \to \mathcal{Y}$. Given an image patch $x \in \mathcal{X}$, f predicts the category of the center pixel (object or background). Inspired by the hypercolumn representation approach used in [28], we describe an algorithm to achieve this in the following.

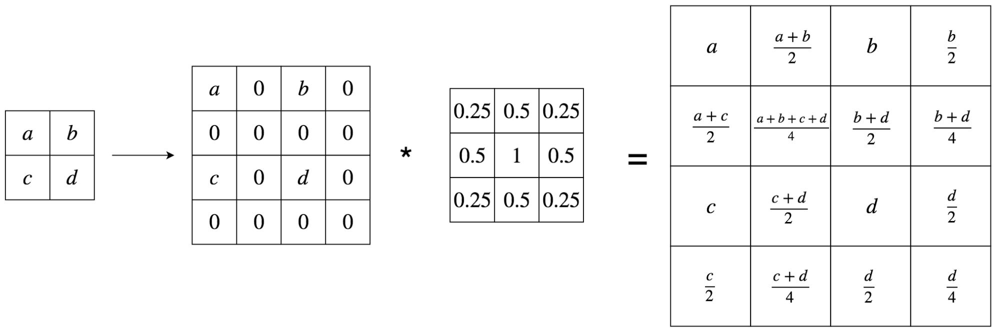


Figure 5.2: A $2\times$ upsampling operation by bilinear kernel convolution.

Similar to [28], we feed the input image to a CNN, and we want to extract the CNN features of different layers to form a hypercolumn descriptor for each pixel location. Due to the subsampling and pooling operations that we adopt in the CNN, the feature maps across different layers might have different sizes compared to the input. Therefore, for feature maps that have different sizes from the input, we resize these feature maps to the size of the input using bilinear interpolation. We implement the bilinear interpolation upsampling using a two-step mechanism: we first upsample the feature map with zero stuffing and then apply a designed filter to achieve bilinear interpolation (see Figure 5.2 for an example of $2\times$ upsampling operation using bilinear interpolation). We take the outputs of all the layers, upscale the feature outputs to have the same resolution with the input using bilinear interpolation, and append all the upscaled feature outputs together to be the hypercolumn descriptor for each pixel.

We can train the entire pipeline end-to-end to predict the binary support masks. During training, we provide the original images and the corresponding binary target maps that have the same resolution as the input images and targets. We can calculate the logistic loss for each pixel location and the final loss can be calculated as the mean of the losses across all the pixel locations in the image. We show the model architecture in Figure 5.3.

Note that we still need to provide synthetic images with real background as the training images together with the binary target support maps as the labels during training. Since the synthetic images are generated from 3D CAD models, it is much easier to create target support maps as we can infer the foreground and background segmentation from the synthetic images. We will discuss the synthetic image generation procedure more in detail in the next section.

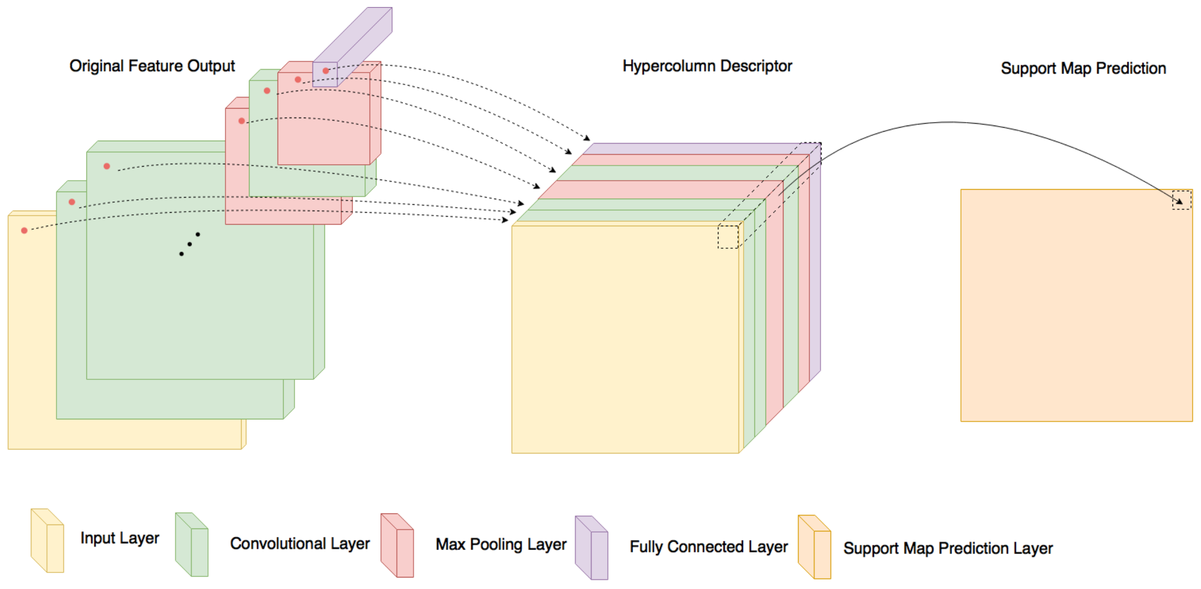


Figure 5.3: End-to-end model architecture for support map prediction.

5.4.3 Discussion

Learning image object extraction is not the goal of this approach; it is rather a preprocessing step for the test images so that we can directly apply the classification model that is trained on synthetic object images with clear background. Therefore, three questions need to be answered: One, for synthetic test images, are we able to get reasonable object support masks by the pixel-wise prediction based on the hypercolumn descriptors? Two, if we use real images as the test images, are we still able to get reasonable object support masks even though the object extraction model is trained on synthetic images? Three, if the answers to the first two questions are yes, then the third question we need to answer is that: Are we able to get good classification performance on real images of extracted target objects (background removed) when the classification model is trained on synthetic object images with clear backround? We will design some experiments to answer these questions.

Similar to the few-shot learning approach we described in Section 5.3, this approach also requires us to generate synthetic images with real background images as the training data. It is therefore interesting to compare the performance between the few-shot learning approach and the approach of classification on the extracted objects. We will discuss the advantage of each approach respectively in the experiment section.

5.5 Generate Synthetic Images from 3D CAD Models

In this section, we describe how to create synthetic images by generating realistic image objects using 3D CAD models and selecting real images as the background. We also explore how we can create object support masks for the created synthetic images.

5.5.1 Render using 3D CAD models

CAD models of different objects are becoming publicly available online. In this work, we use the ShapeNet dataset [13], a large-scale dataset of 3D shapes that covers 55 common object categories with more than 50,000 3D models. It is also organized according to the WordNet hierarchy [53], making it easier to connect with some existing 2D image datasets such as the ImageNet dataset [67]. With the available 3D models, we use a publicly available OpenGL-based renderer to create the object images based on the 3D models.

When rendering the object images, we consider the following factors: the horizontal and vertical axis of rotation (both control the spatial rotation of the object) and the distance between the viewpoint and the object (controls the size of the rendered object). We set a proper range for each of these factors to make sure the rendered images look reasonable. To create random views of an object, we set random values within the proper ranges for these factors and pass them to the rendering program. In addition to that, we create six canonical views of the objects including the top, bottom, left, right, front, and back views while properly setting the distance so that the entire model fits within the canvas.

In Figure 5.4, we show six canonical views of an object as well as some randomly generated views of the same object. As we will discuss in the following, these generated views of objects will be combined with selected background images to create the synthetic images.

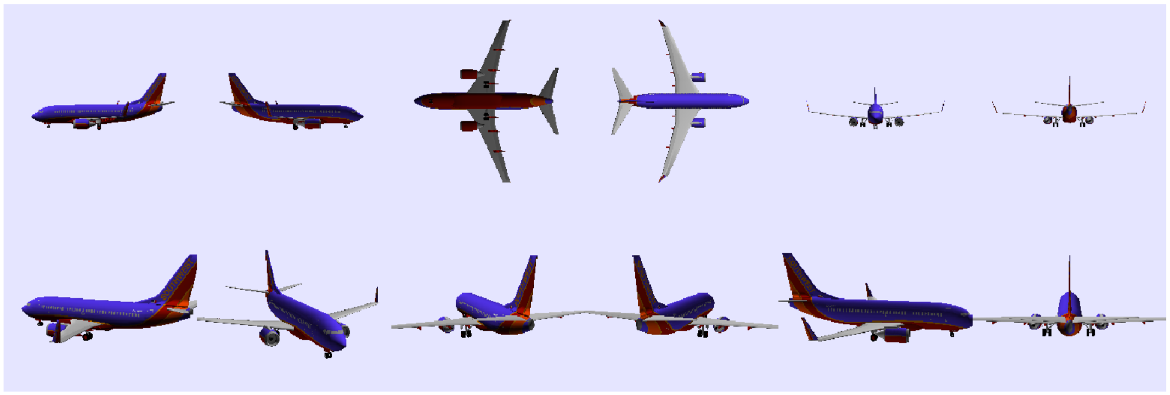


Figure 5.4: Six canonical rendered views (first column) along with randomly generated views (second column) of a CAD plane model.

5.5.2 *Generate Support Maps for 3D objects*

As we discussed above, one direction that we are interested in is to create a model to generate support masks for the target objects in the real images. In order to acquire enough data to train such a model for real images, it requires lots of human labor to annotate the real images. One of the advantages of generating synthetic images is that we can achieve the support masks for the objects without extra effort. When generating the object images, we set a unique background color that is different from any texture color, and we are able to get the support masks by capturing the regions that do not have the background color. In Figure 5.5, for example, we show the support masks for the objects in the synthetic images.

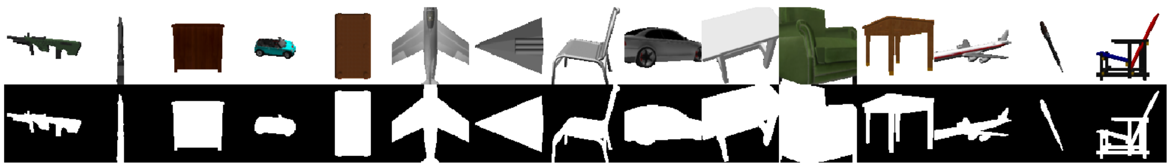


Figure 5.5: Rendered images with clear background (first column) and their corresponding support masks (second column).

5.5.3 Background in Synethetic Images

A realistic object appearance also depends on the background information. Here we describe how to create background images and then put the rendered image objects in front of the background to generate the synthetic images.

One source of background images for the synthetic objects are real images of different scenes we collected as the background images. We download some scene images from the Google Image website using search keyword such as “sky”, “road”, “sea”, “room”, etc. Such images downloaded from the Google Image website usually have been post-processed, and some important image statistics such as saturation, chroma, and contrast have been carefully tuned to make them more appealing. In order to recover these processes and bring them closer to natural images, we use the cylinder-coordinate HSV representation of the color information and adjust the hue coordinate and the value coordinate to match the image statistics with those of the natural images. Another source of background images is the Flickr 100k dataset, which consists of 100071 images. For those, we do not need to apply any post-processing procedure and they can be used directly as the background images for our image objects.

It is undeniable that there are some correlations between the object classes and their background: planes and cars usually have outdoor scenes like sky and roads, whereas furniture usually appears in indoor scenes. In our work, we try to eliminate these correlations when selecting background images for the objects and allow images objects to appear in some unrealistic scenes (e.g., planes inside a bedroom).

After we select the background images, we need to combine the object images and the background images to create synthetic images of the target objects. As we already have the support maps, we can replace the non-background regions of the background images with the some object images to create synthetic images. Note that some images are selected from the Flickr dataset using some popular tags, including tags of some real objects like “animals”, "car", "cat", "baby" and tags of things that are not objects like "blue", "summer", "uk" and "spain". Therefore, some of these images may already contain an object in the scene, which makes it harder to discriminate the target objects in the foreground from the noisy background. In Figure 5.6, we show the synthetic images generated by putting rendered 3D CAD models in front of different types of real background images.

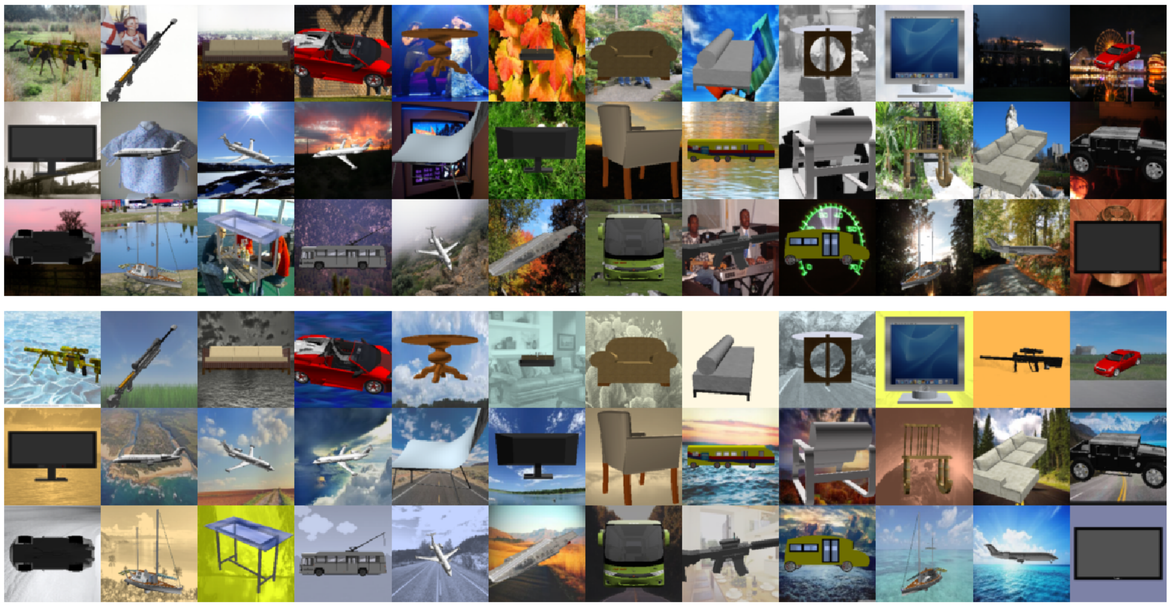


Figure 5.6: Examples of synthetic images by combining the rendered 3D objects and real image background. The background images of these synthetic images come from the Flickr dataset (top) and the Google Image website (bottom) respectively. As we can observe from these images, background images from the Flickr dataset usually contain other objects, which make the background noisier than the images we collected from the Google Image website.

5.6 Experiments

In order to explore the effectiveness of the discussed approaches when applied to real images, we perform classification experiments on the CIFAR-10 dataset and the ImageNet dataset. Note that since we only have CAD models for a few object categories, we only select images of the available objects categories to form subsets of the original datasets. In our case, we create a CIFAR-10 3-class dataset and an ImageNet 9-class dataset, and we evaluate the performance of our approaches on these two datasets.

5.6.1 CIFAR-10 3-Class Dataset

In this experiment, we want to see how the synthetic images generated by 3D CAD models can help classify the object images from the CIFAR-10 dataset. As we discussed earlier, we render synthetic images using the 3D CAD models from the ShapeNet dataset, which contains more than 50,000 3D models from 55 common object categories. Among these 55 object categories, three object categories also exist in the CIFAR-10 dataset, including "plane", "boat", and "car" . Therefore, we collect the CIFAR-10 images that are from these three classes and perform a three-class classification task. The training and testing data partition is the same as the original CIFAR-10 dataset. We then follow the process described in Section 5.5 to generate synthetic images with background. For each of the three classes, we collect 30,000 synthetic images by rendering 3D CAD models using randomly selected camera positions, and we use the first 20,000 images per class as the synthetic training data and the rest 10,000 images as the synthetic testing data.

To solve the classification task, we build a deep CNN as a classifier and the CNN architecture is adapted from the VGG-16 model that has shown reasonably good classification performance on the CIFAR-10 dataset. The detail of the model architecture is shown in Figure 5.1. We first see how well the model can classify the CIFAR-10 3-class test dataset when we augment the CIFAR-10 training data with the synthetic data. We show the experiment results in Table 5.1. By augmenting the CIFAR-10 training data using the synthetic images, we observe significant improvements on the performances of few-shot learning tasks. When more and more training data from the CIFAR-10 training dataset is available, we can see that the improvement from using synthetic images becomes smaller.

Then we take another approach, which is to extract the target objects in the real images

Number of training data from CIFAR-10	With Synthetic Images	Without Synthetic Images
0 per class	76.23%	N/A
100 per class	83.97%	66.70%
1000 per class	91.80%	89.23%
5000 per class	95.50 %	95.07%

Table 5.1: Classification accuracy on CIFAR-10 3-class test data when limited amount of real training data is augmented by synthetic data.

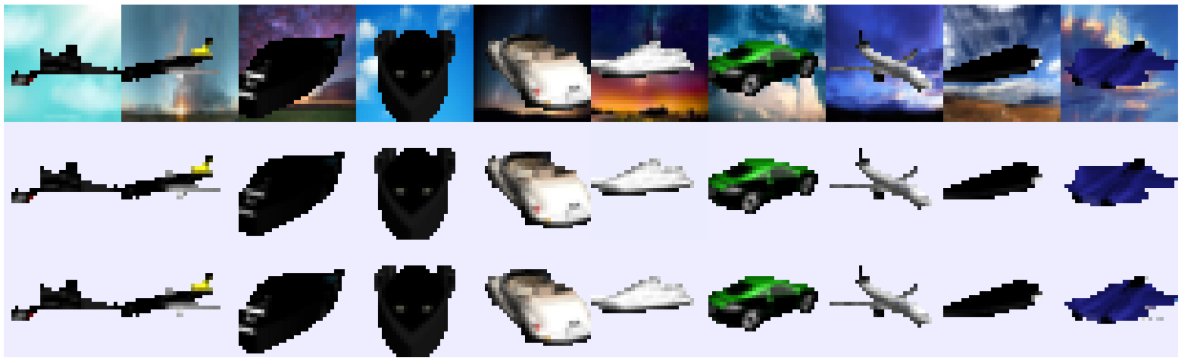


Figure 5.7: We show the synthetic image (the first row) and the corresponding target objects in the image (the second row). We show in the third row the extracted target object produced by applying the support maps to the original synthetic image.

and remove the background before classifying the images. To see why this is helpful, we first apply the same procedure to the synthetic test images and investigate the classification experiments on the synthetic images. We first create synthetic images with real image background as the training data along with the corresponding support maps as the labels. A pixel-wise classification model is then trained on the synthetic data, and it is used to find the support masks of the target objects and remove the background in the synthetic test images. In Figure 5.7, we show example images of extracted target objects by applying the support masks for the synthetic images using the pixel classification model. As we can see from the figure, the algorithm does a good job extracting the target objects from the synthetic images, and the extracted target objects are almost identical with the ground-

	Synthetic test images	Synthetic test images (background removed)	Synthetic test image without background
Train on synthetic images without background	92.29%	95.0 %	96.43%
Train on synthetic images with background	93.57%	95.5%	96.61%

Table 5.2: Classification result on different types of synthetic test images.

truth objects in the images. After we extract the target objects, we can directly apply a classification model that is trained on synthetic object images without background to classify the synthetic images of extracted objects. In Table 5.2, we show that by removing the background, classification accuracy on the synthetic test images is improved from 92.29% to 95.0% when the classification model is trained on the synthetic images without background, and it is also improved from 93.57% to 95.5% when the model is trained on the synthetic images with real background. This shows the advantage of removing image background before classifying the images. We show the accuracy rates when tested on the synthetic images without background on the third column. Note that if the pixel classification model can perfectly extract the target objects from the images, the accuracy rates on the second column should be the same with the third column.

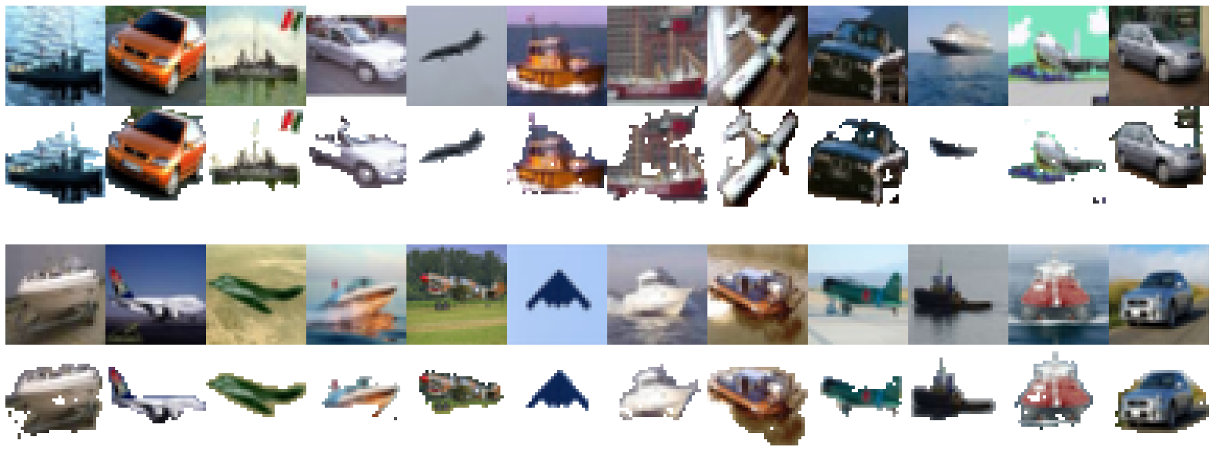


Figure 5.8: We show some CIFAR-10 images examples and the corresponding extracted target objects produced by applying the support maps to the original images.

We then apply the same procedure to classify the real test images from the CIFAR-10 3-class test dataset. Both the pixel classification model and the image classification model are still trained using the synthetic images, but this time we will test the models on the real images from the CIFAR-10 3-class test dataset. In Figure 5.8, we show example images of the extracted target objects by applying the support masks for the real images using the pixel classification model. As we can see from the figure, although the extracted objects are not as clean as shown in Figure 5.7 for the synthetic images, the model is still doing a reasonably good job extracting the target objects from the real images. Of course, some extracted objects are misleading due to the poor support masks generated by the algorithm, and these objects can be mistaken for a different category. For instance, in the 10th example of the top two rows in Figure 5.8, since the support mask does not cover the entire body of the ship, it could be misclassified as an airplane if we extract the target object by applying this support mask to the original image.

Since we are able to extract target objects from the real test images, we can classify these objects using a classification model that is trained on the synthetic images of objects without background. In Table 5.3, we show the classification accuracies on the CIFAR-10 3-class test data when we apply different training and testing mechanisms. When the classification model is trained on the synthetic images without background, we observe a clear improvement of performance by removing image background before classifying the test images, as the model trained on the images of objects with clear background is certainly not

Training Data	Original Test Images (with background)	Original Test Images (background removed)
Synthetic images without background	41.83%	63.60%
Synthetic images with background	76.39%	58.43%

Table 5.3: Classification accuracies on CIFAR-10 3-class test data.

robust to the noisy background in the test images. On the other hand, when the model is trained on the synthetic images with real background, we observe performance degradation when an object-extraction procedure is applied before classifying the test images. This is expected due to the noise introduced when extracting target objects from the test images, and the model trained on synthetic images with background is not able to achieve good performance when the extracted objects from test images are not clear enough. Also, we notice that the best classification accuracy is achieved when we train the model using the synthetic images with real background and test it on the original test images. This is different from what we observed from the experiment results in Table 5.2 where the best performance on the synthetic images with real background is achieved when we apply the object-extraction procedure before classifying the test images. It is possible that the images of the extracted objects from the real images are not as clear as those extracted from the synthetic images as we showed in Figure 5.8, and it leads to poor performance when applying the object-extraction procedure before classifying the images.

5.6.2 Downsampled ImageNet Dataset

In order to show the generality of our approach, we apply our approach to the downsampled ImageNet dataset. Similar to the experiments in the last section, we use the 3D CAD models from the ShapeNet dataset to create synthetic images. Each object category among the 55 object categories in the ShapeNet dataset has a unique WordNet ID, and a corresponding subset of the images from the ImageNet dataset can be accessed using the same WordNet ID. We select nine object categories (including boat, bus, car, chair, display, gun, plane, sofa, and table) that have sufficient models in the ShapeNet dataset, and we collect the corresponding images from the 32×32 downsampled ImageNet training dataset to form an ImageNet 9-class dataset. For each category in the ImageNet dataset, we collect 1200 images where the first 1000 images are used for training and validation, and the rest 200 images

Number of training data from the ImageNet 9-class training set	With Synthetic Images	Without Synthetic Images
0	45.56%	N/A
100 per class	59.28%	45.28%
500 per class	70.67%	61.22%
1000 per class	75.17%	68.06%

Table 5.4: Classification accuracy on the ImageNet 9-class test data when synthetic images are created to augment the real training images.

are used for testing. We also collect 30,000 synthetic images for each class by rendering 3D CAD models using randomly selected camera positions, where the first 20,000 images per class are used as training data, and the rest 10,000 images are used as testing data.

Similar to our previous experiment, we first explore how synthetic data augmentation approach can help improve the classification accuracy on the ImageNet dataset. We build a deep CNN model for the classification task, and the architecture of the model is shown in Figure 5.1. In Table 5.4, we observe significant performance improvement in the few-shot learning experiments when we augment the training data with synthetic images. This is consistent with what we have seen in the previous experiments.

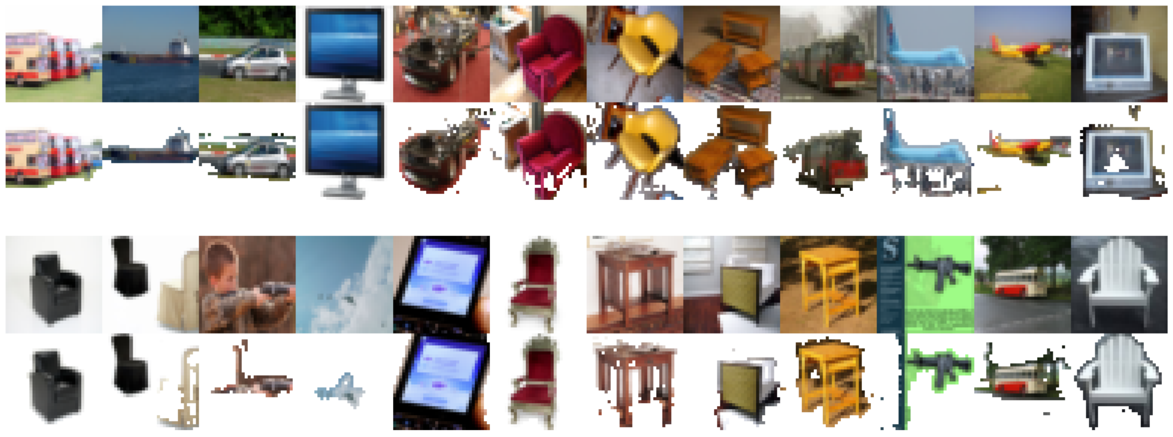


Figure 5.9: We show some example ImageNet images and the corresponding extracted target objects produced by applying the support maps to the original images.

Training Data	Original Test Images (with background)	Original Test Images (background removed)
Synthetic images without background	16.44%	29.22%
Synthetic images with background	45.56%	36.11%

Table 5.5: Classification accuracies on the ImageNet 9-class test data.

We then apply the object-extraction procedure to the ImageNet 9-class test images before classifying them. Similarly, a pixel classification model and an image classification model are trained using synthetic images. We first show how well the pixel classification model can extract the target objects from the real test images and remove the background. In Figure 5.9, we show example images of the extracted target objects by applying the support maps for the real images from the ImageNet 9-class test dataset. We can then classify the images of extracted objects using a classification model trained on the synthetic images of objects. In Table 5.5, we observe a similar performance pattern compared to the experiment results we presented in Table 5.3: A significant performance improvement is achieved by applying the object-extraction procedure before classifying the images when the classification model is trained on the synthetic images without background. As is discussed above, it is also not surprise to observe a poorer result if we extract the objects from images before classifying them when the classification model is trained on the synthetic images with real background. The overall best accuracy rate is achieved when the model is trained on the synthetic images with background and directly tested on the original test images without the object-extraction procedure.

5.7 Conclusion

In this work, we study possible methods to take advantage of 3D CAD models to explore the transformation variations of the image objects that are not easily available in the existing

2D image dataset. Data augmentation with synthetic images rendered from 3D CAD models allows the model to observe data variations that do not exist in the training data, and it shows significant performance improvement for the classification tasks especially when we have zero or few training examples.

We also show a foreground-background segmentation model trained on synthetic images can reasonably extract the target objects from the real images. If the target objects can be correctly extracted from the real images and the noisy background of the images are completely removed, then a classification model trained on synthetic images with clear background can be directly applied to classifying the images of extracted objects. We observe a significant performance improvement by applying the object-extraction procedure before classifying the images when the classification model is trained on synthetic images without background. Performance degradation is also observed if we perform classification task on the extracted object when the classification model is trained on synthetic images with background. We argue, however, that if such a segmentation model can perfectly extract the target objects from the real images, applying the object-extraction procedure should improve the classification result despite the type of training data, as we have already shown in the synthetic image classification experiment in Table 5.2.

REFERENCES

[1] Yali Amit. *2D object detection and recognition: Models, algorithms, and networks*. MIT Press, 2002.

[2] Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. *Advances in neural information processing systems*, pages 577–584, 2003.

[3] Mathieu Aubry, Daniel Maturana, Alexei A Efros, Bryan C Russell, and Josef Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3762–3769, 2014.

[4] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[6] Elliot Joel Bernstein and Yali Amit. Part-based statistical models for object classification and detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 734–740. IEEE, 2005.

[7] Manuel Blum, Jost Tobias Springenberg, Jan Wülfing, and Martin Riedmiller. A learned feature descriptor for object recognition in rgb-d data. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1298–1303. IEEE, 2012.

[8] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Unsupervised feature learning for rgb-d based object recognition. In *Experimental Robotics*, pages 387–402. Springer, 2013.

[9] Fred 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.

[10] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In *European conference on computer vision*, pages 536–551. Springer, 2014.

[11] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016.

[12] Joao Carreira and Cristian Sminchisescu. Cpmc: Automatic object segmentation using constrained parametric min-cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1312–1328, 2012.

[13] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[14] Gong Cheng, Peicheng Zhou, and Junwei Han. Rifd-cnn: Rotation-invariant and fisher discriminative convolutional neural networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2884–2893, 2016.

[15] Taco S Cohen and Max Welling. Group equivariant convolutional networks. *arXiv preprint arXiv:1602.07576*, 2016.

[16] Taco S Cohen and Max Welling. Steerable cnns. *arXiv preprint arXiv:1612.08498*, 2016.

[17] Sander Dieleman, Jeffrey De Fauw, and Koray Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. *arXiv preprint arXiv:1602.02660*, 2016.

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

[19] Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 766–774, 2014.

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

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

[22] Robert Gens and Pedro M Domingos. Deep symmetry networks. In *Advances in neural information processing systems*, pages 2537–2545, 2014.

[23] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[24] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[25] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *European Conference on Computer Vision*, pages 345–360. Springer, 2014.

[26] Saurabh Gupta, Judy Hoffman, and Jitendra Malik. Cross modal distillation for supervision transfer. *arXiv preprint arXiv:1507.00448*, 2015.

[27] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision*, pages 297–312. Springer, 2014.

[28] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, 2015.

[29] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.

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

[31] João F Henriques, Pedro Martins, Rui F Caseiro, and Jorge Batista. Fast training of pose detectors in the fourier domain. In *Advances in neural information processing systems*, pages 3050–3058, 2014.

[32] João F Henriques and Andrea Vedaldi. Warped convolutions: Efficient invariance to spatial transformations. *arXiv preprint arXiv:1609.04382*, 2016.

[33] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In the 12th International Symposium on Experimental Robotics (ISER*. Citeseer, 2010.

[34] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888, 2012.

[35] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer, 2012.

[36] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[37] Stefan Holzer, Jamie Shotton, and Pushmeet Kohli. Learning to efficiently detect repeatable interest points in depth data. In *European Conference on Computer Vision*, pages 200–213. Springer, 2012.

[38] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015.

[39] Edward Johns, Stefan Leutenegger, and Andrew J Davison. Pairwise decomposition of image sequences for active multi-view recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3813–3822, 2016.

[40] Koray Kavukcuoglu, Rob Fergus, Yann LeCun, et al. Learning invariant features through topographic filter maps. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1605–1612. IEEE, 2009.

[41] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. *arXiv preprint arXiv:1704.01222*, 2017.

[42] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

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

[44] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE, 2011.

[45] Dmitry Laptev, Nikolay Savinov, Joachim M Buhmann, and Marc Pollefeys. Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 289–297, 2016.

[46] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM, 2007.

[47] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[48] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 991–999, 2015.

[49] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[50] Diego Marcos, Michele Volpi, Nikos Komodakis, and Devis Tuia. Rotation equivariant vector field networks. *arXiv preprint arXiv:1612.09346*, 2016.

[51] Javier Marin, David Vázquez, David Gerónimo, and Antonio M López. Learning appearance in virtual scenarios for pedestrian detection. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 137–144. IEEE, 2010.

[52] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[53] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[54] Lian Huan Ng, Gustav Larsson, Jiajun Shen, and Yali Amit. Stacked statistical models with rotatable features. In *Technical report*, 2014.

[55] Jiquan Ngiam, Zhenghao Chen, Daniel Chia, Pang W Koh, Quoc V Le, and Andrew Y Ng. Tiled convolutional neural networks. In *Advances in neural information processing systems*, pages 1279–1287, 2010.

[56] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1520–1528, 2015.

[57] Panagiotis Papadakis, Ioannis Pratikakis, Theoharis Theoharis, and Stavros Perantonis. Panorama: A 3d shape descriptor based on panoramic views for unsupervised 3d object retrieval. *International Journal of Computer Vision*, 89(2):177–192, 2010.

[58] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning deep object detectors from 3d models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1278–1286, 2015.

[59] Bojan Pepik, Rodrigo Benenson, Tobias Ritschel, and Bernt Schiele. What is holding back convnets for detection? In *German Conference on Pattern Recognition*, pages 517–528. Springer, 2015.

[60] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Pro-*

ceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5648–5656, 2016.

[61] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[62] Siamak Ravanbakhsh, Jeff Schneider, and Barnabas Poczos. Deep learning with sets and point clouds. *arXiv preprint arXiv:1611.04500*, 2016.

[63] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[64] Gernot Riegler, Ali Osman Ulusoys, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. *arXiv preprint arXiv:1611.05009*, 2016.

[65] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[66] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[67] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[68] Nima Sedaghat, Mohammadreza Zolfaghari, and Thomas Brox. Orientation-boosted voxel nets for 3d object recognition. *arXiv preprint arXiv:1604.03351*, 2016.

[69] Konstantinos Sfikas, Theoharis Theoharis, and Ioannis Pratikakis. Exploiting the panorama representation for convolutional neural network classification and retrieval. In *Eurographics Workshop on 3D Object Retrieval, Lyon, France*, 2017.

[70] Baoguang Shi, Song Bai, Zhichao Zhou, and Xiang Bai. Deeppano: Deep panoramic representation for 3-d shape recognition. *IEEE Signal Processing Letters*, 22(12):2339–2343, 2015.

[71] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[72] Kihyuk Sohn and Honglak Lee. Learning invariant representations with local transformations. *arXiv preprint arXiv:1206.6418*, 2012.

[73] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[74] Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2686–2694, 2015.

[75] Damien Teney and Martial Hebert. Learning to extract motion from videos in convolutional neural networks. *arXiv preprint arXiv:1601.07532*, 2016.

[76] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

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

[78] Fa Wu, Peijun Hu, and Dexing Kong. Flip-rotate-pooling convolution and split dropout on convolution neural networks for image classification. *arXiv preprint arXiv:1507.08754*, 2015.

[79] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015.

[80] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2014.

[81] Jiaolong Xu, David Vázquez, Antonio M López, Javier Marín, and Daniel Ponsa. Learning a part-based pedestrian detector in a virtual world. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):2121–2131, 2014.

[82] Yanzhao Zhou, Qixiang Ye, Qiang Qiu, and Jianbin Jiao. Oriented response networks. *arXiv preprint arXiv:1701.01833*, 2017.