Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's project

Geometry and Transformations in Deep Convolutional Neural Networks

Tomáš Jakab

Supervisor: Dr Andrea Vedaldi, University of Oxford

Study Programme: Open Informatics

Field of Study: Computer and Information Science

May 27, 2016

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Tomáš J a k a b

Study programme: Open Informatics

Specialisation: Computer and Information Science

Title of Bachelor Project: Geometry and Transformations in Deep Convolutional Neural Networks

Guidelines:

1. Familiarize yourself with recent research studying handling of geometry in deep convolutional neural networks.
2. Propose experiments evaluating the ability of deep representations to handle geometric information, in order to support transformation-invariant image recognition. Such experiments can involve identity recognition of transformed objects where information about object geometry should be preserved through the network to allow successful recognition.
3. Create dataset for proposed experiments.
4. Depending on the results of the previous investigation, propose improvements of current architectures and evaluate their contributions.

Bibliography/Sources:

[1] Lenc K, Vedaldi A. Understanding image representations by measuring their equivariance and equivalence; The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015

[2] Yang J, Reed SE, Yang MH, Lee H. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. Advances in Neural Information Processing Systems, 2015

Bachelor Project Supervisor: Andrea Vedaldi, Ph.D.

Valid until: the end of the summer semester of academic year 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 4, 2016

Author statement for undergraduate thesis

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accoradance with the methodical instructions for observing the ethical principles in the preparation of university thesis.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

In Prague on May 27, 2016 .....

Aknowledgements

I would like to express my deepest gratitude to my supervisor, Dr Andrea Vedaldi, for his support and guidance throughout the work on this project. I also wish to thank Prof Jiří Matas for his support and valuable advice he gave me during my studies. Finally, I would like to express my special appreciation to my parents and grandparents for their love and continued support.

Abstract

In this work, we investigate how the classification error of deep convolutional neural networks (CNNs) used for image verification depends on transformations between two visually similar inputs. Furthermore, inspired by research in experimental psychology suggesting that humans solve such a task by mentally rotating the observed objects, we test whether building such a mechanism into artificial neural networks improves their performance. For transformations, we consider 2D rotations and translations. We show that, in the case of rotation, the lowest error is achieved when the two inputs are aligned, followed by two minima for rotations of 90 and 180 degrees, although the difference is small for standard architectures. The difference is much more pronounced when "mental rotations", implemented by means of resampling and spatial transformer layers, are introduced in the architecture, significantly improving the performance of the vanilla architecture. Finally, we qualitatively probe deep features using a method for feature inversion. We discovered invariant properties of the first fully connected layer (FC1) features adapted to input rotation. Our results also suggest that in networks trained to predict rotation between two inputs, the FC1 features implicitly learn to establish correspondences between corner points in the rotated inputs.

Keywords: convolutional neural network; CNN; geometry; geometric transformation; rotation; translation; invariance; image verification; correspondence

Abstrakt

Tato práce zkoumá jak klasifikační chyba hlubokých konvolučních sítí (CNN) používaných pro verifikaci obrázků závisí na transformaci mezi dvěma vizuálně podobnými obrázky. Inspirování výzkumem v experimentální psychologii, který se domnívá, že lidé řeší takovéto úlohy pomocí mentální rotace pozorovaných objektů, chceme ověřit, jestli zabudování takovéhoto mechanismu do neuronových sítí vylepší jejich klasifikační přesnost. Kontréně uvažujeme 2D rotace a translace. V této práci ukážeme, že v případě rotace je nejnižší chybovost dosažena, pokud jsou oba obrázky zarovnány. Další minima jsou při 90 a 180 stupních rotace mezi dvěma vstupy. Nejnižší chybovost v připadě žádné rotace je navíc podpořena dalšími výsledky ukazujícími, že použití mechanismu „mentální rotace" před tím, než jsou obrázky poslány do verifikační neuronové sítě, výrazně zlepší klasifikační přesnost jednoduché CNN. Nakonec kvalitativně vyšetříme hluboké reprezentace za použití metod pro invereze reprezentací. Díky tomu jsme objevili invariantní vlastnosti reprezentací z první plně propojené vrstvy (FC1), která se adaptovala na rotace vstupního obrázku. Na základě našich dalších výsledků se domníváme, že v sítích trénovaných pro predikci rotace mezi dvěma obrázky, se FC1 reprezentace implicitně naučí korespondece mezi dvěma rotovanými obrázky.

Klíčová slova: konvoluční neuronové sítě; geometrie; geometrické transformace; rotace; translace; invariance; verifikace obrázků; korespondece

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Deep convolutional neural networks (CNN) are extremely powerful for certain tasks such as classification. However, they process data in a feed forward manner and it is unclear whether they can solve image analysis problems that require more careful processing. In psychology there is a similar distinction as some tasks need more time to be solved. A prime example is deciding whether two visual objects are the same up to two or three dimensional transformations, such as the ones induced in a viewpoint change. Shepard & Metzler [19] presented their subjects with 2D drawings of pairs of 3D objects and asked them to decide whether the images contains the same object or its reflection. Their study showed that reaction time of the participants was proportional to the angular rotation between the presented objects. Cooper [3] did similar experiments with random polygons confirming that time needed to solve the verification task depends on the rotation. Thus, the more rotation is between two objects the more difficult the verification is.

Inspired by this, we examine whether similar phenomena occurs also in deep neural networks. Moreover, since both of the studies hypothesized that humans solve such a task by mentally rotating the observed objects in order to align them, we investigate whether mental rotation can be also useful for CNNs. For this purpose, we consider a number of problems that consist of matching objects up to transformations of the image. We decided to use a simple type of synthetic data reminiscent of the one used by [3] or [7]. As our goal is to verify whether networks have the ability of representing correctly certain factors of variation, synthetic data are very useful for controlling such factors perfectly. More specifically, we want to have subtle visual differences between different classes, but relatively large differences within classes caused by geometric transformations. This would test network ability to cope with large viewpoint changes, while the network would have to also retain important information about the object geometry. Although our primary interest is in 2D rotation, we also provide comparison with 2D translation. Finally, we examine learned features of the networks used and their changes under the input image transformations.

In the first chapter we discuss our motivation and related literature. The second chapter provides overview of basic building blocks presented in current CNNs and describes architectures we used in our experiments. The third chapter discusses generation of synthetic

datasets. The fourth chapter summarizes our experiments and results. The fifth chapter discusses features visualization and analysis. Finally, the last chapter provides a summary of this work.

1.2 Related literature

In this section we discuss the work related to the thesis. However, there is limited work on understanding geometry and transformations in CNNs. Literature related to understanding mostly focuses on various visualization techniques [15, 5, 20, 25], some of those works are later discussed in the 4th and 5th chapter. Probably the most relevant work is by Lenc & Vedaldi [13] studying invariance and equivariance properties of CNN representations by finding linear transformations of the learned features in convolutional layers. Work by Aubry & Russell [1] analyzed the variation of CNN features with respect to various scene factors such as change of viewpoint. In contrast to our work, both of the works used complex models that were already pre-trained on the classification task of the ILSVRC-2013 challenge [4]. Cohen & Welling [2] analyze transformation properties of representations using theory of symmetry groups. Memisevic & Hinton [18, 16, 17] studied learning relations between transformed images using RBMs. In contrast to standard deep learning models, they use three-way multiplicative interactions. This means that value of a variable depends on a product of other variables. The rest of literature related to transformations and geometry focuses on building transformation invariant mechanisms into the CNNs. However, we want to focus on mostly vanilla CNNs.

Chapter 2

Deep network architectures

In this chapter we firstly provide a brief introduction to CNNs describing their basic building blocks. Secondly, we describe architectures that we used in experiments.

2.1 CNN overview and its basic building blocks

A neural network is a function $\mathbf{y} = g(\mathbf{x}; \mathbf{w})$ parametrized by the vector $\mathbf{w}$ that maps the input vector $\mathbf{x}$ to the output vector $\mathbf{y}$. The function g can be thought of as the composition $g = f_N \circ \cdots \circ f_1$ of simpler functions f_n called layers. The functions f_n have to be at least piecewise continuously differentiable in order to be able to optimize the neural network with gradient based methods.

CNNs were first introduced in 1998 by LeCun at al. [12], however, the basic idea inspired by mammalian visual cortex [9] was presented much earlier in 1982 by Fukushima at al. [8]. The work from 2012 by Krizhevsky at al. [11] marks beginning of convolutional network renaissance in computer vision and many of the current architectures are iterations of the one presented in their work. CNNs are nowadays widely used in various computer vision tasks such as classification [21], localization [20], semantic segmentation [14, 26], surface normals estimation [6].

CNNs leverage spatial structure in the data, which is represented by rank-3 tensor $\mathbf{x} \in \mathbb{R}^{H \times W \times D}$, where H is the height, W is the width, and D is the number of feature channels. In computer vision, H and W are 2D spatial dimensions. The features channels in the input tensor can represent color channels and/or depth channel. However, CNNs can be generalized to any tensor rank. For example, a rank-4 input tensor can represent volumetric data, which is commonly used in medical imaging.

2.1.1 Basic building blocks

In this section, we provide an overview of basic functional blocks frequently used in current CNNs. Those are convolution, non-linear gating, pooling, softmax, and loss layer.

2.1.1.1 Convolutional layer

The convolutional layer is a function parametrized by a tensor $\mathbf{w} \in \mathbb{R}^{H \times W \times D \times K}$, where H is the height, W is the width, D is the filter depth, and K is the number of filters. The tensor $\mathbf{w}$ is called the filter bank. The convolutional layer takes an input tensor $\mathbf{x} \in \mathbb{R}^{H \times W \times D}$ and convolves it across its height and width with each of the filters from the filter bank $\mathbf{w}$ producing an output tensor $\mathbf{x}' \in \mathbb{R}^{H' \times W' \times K}$. The layer can be furthermore parametrized by hyper-parameters — padding p and stride s. Padding pads the spatial dimensions of the input tensor with zeros. Stride s of size higher than 1 down-samples the output tensor taking every sth point across its spatial dimensions.

2.1.1.2 Fully-connected layer

The fully-connected layer is a special type of convolutional layer that has a filter bank with the same size of the spatial dimensions as the input tensor has. Therefore, every point in the output data depends on every point from the input data.

2.1.1.3 Non-linear gating layer

The non-linear gating layer applies a non-linear function, called activation function, on every point from the input tensor. The most common function

$$f(x) = \max(0, x)$$

is called a rectified linear unit, abbreviated as ReLU. Other functions can be also used. For example the sigmoid function

$$f(x) = \frac{1}{(1 + \exp^{-x})}.$$

2.1.1.4 Pooling layer

The pooling layer is a form of down-sampling function. Its role in CNNs is to reduce the spatial dimensions of the data, while attaining invariance to small deformations by recording the strongest activation of features in small neighbourhoods. The poling layer is specified by its filter size $H \times W$, padding p, stride s (these two parameters have the same role as in the convolutional layer), and a pooling function. The layer acts as a convolutional layer, but instead of computing dot-product it applies the pooling function. The max-pooling, which is the most common one, takes the maximum of the subregion within the filter. Another type of pooling function can be for example average pooling that computes the average.

2.1.1.5 Softmax layer

The softmax layer is a normalization function $f : \mathbb{R}^{H \times W \times D} \to \mathbb{R}^{H \times W \times 1}$ that applies the softmax operator

$$y_{ijk} = \frac{e^{x_{ijk}}}{\sum_{l=1}^{D} e^{x_{ijl}}}$$

on the data $\mathbf{x}$ at each spatial location along a column of features. The resulting values can be thought of as posterior probabilities.

2.1.1.6 Loss layer

The loss layer is a loss function that estimates how far the prediction of the network is from the desired output, as dictated by the ground truth-labels. When training the network, the loss function is the last layer, i.e. the outermost function in the function composition. Our goal is to find such a parametrization of the neural network so the loss function is minimized. In case of a classification task the log loss function is used. The log loss function $l(\mathbf{x}; \mathbf{c}) = -\operatorname{vec}(\mathbf{c})^T \log(\operatorname{vec}(\mathbf{x}))$ takes the tensor $\mathbf{x}$ of predicted posterior probabilities and the tensor $\mathbf{c}$ of ground-truth probabilities.

2.1.2 Siamese networks

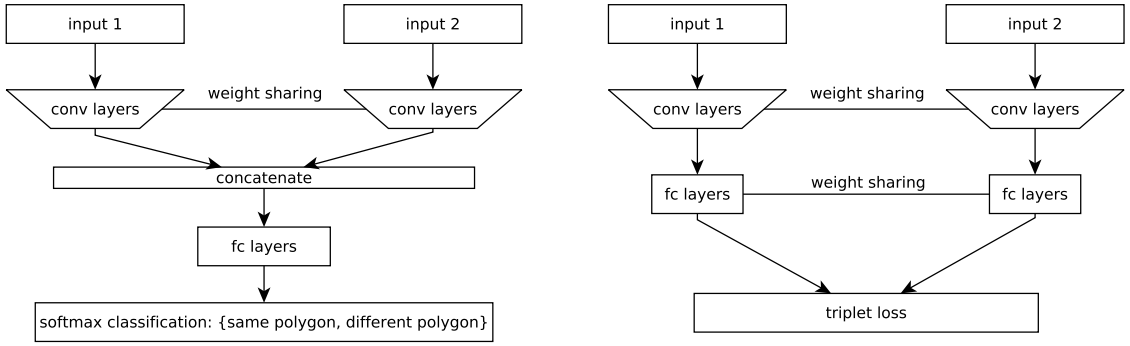


Figure 2.1: **Examples of siamese architecture.**

A Siamese network is a special type of neural network that consists of two identical sub-networks that share their weights. It takes a pair of datums as an input and processes them individually, each in its own subnetwork. After the first n layers the sub-subnetworks are concatenated into a single-stream network or directly fed into a loss layer, see Figure 2.1 for illustration.

2.1.3 Spatial transformer networks

A spatial transformer module [10] performs a spatial transformation of the input features, which the transformation is conditioned on. The module is differentiable and can be directly incorporated into the neural network, which can be trained end-to-end. The spatial transformer takes the input feature tensor $U \in \mathbb{R}^{H \times W \times C}$, which is firstly passed to the localization net $\theta = \phi_{loc}(U)$ predicting parameters θ of the transformation $\mathcal{T}_\theta$, which transforms the regular spatial grid G to the sampling grid $\mathcal{T}_\theta(G)$. Finally, the output feature map $V \in \mathbb{R}^{H \times W \times C}$ is sampled from the input U using the sampling grid $\mathcal{T}_\theta(G)$. The overview of this procedure is provided in Figure 2.2.

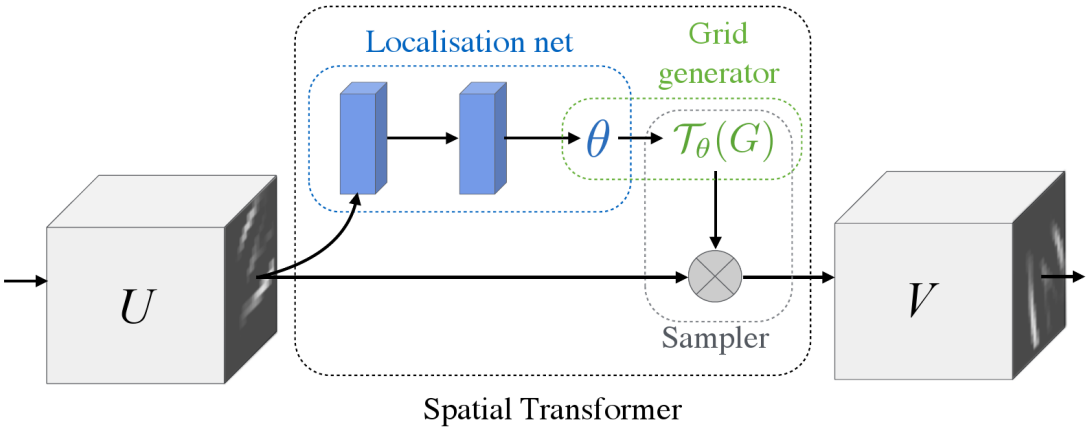


Figure 2.2: **Overview of the spatial transformer module.** U is an input feature map, θ are parameters of the transformation predicted by the localisation network. $\mathcal{T}_\theta(G)$ is the sampling grid where G denotes a regular spatial grid. The image is taken from [10].

Table 2.1: **Baseline architecture.** The table does not show the other identical subnetwork. Note that the depth of the filters in the first fully-connected layer is double the number of filters in the last convolutional layer. This is because the subnetworks are concatenated.

layer type	filter size	stride	padding
conv	5x5x1x32	1	2
ReLU			
maxpool	2x2	2	0
conv	5x5x32x32	1	2
ReLU			
maxpool	2x2	2	0
conv	5x5x32x32	1	2
maxpool	2x2	2	0
ReLU			
concat			
fc	7x7x64x128		
ReLU			
fc	1x1x128x2		
softmax			

2.2 Architecture specification

Because we use very similar architectures throughout the experiments, we describe the baseline architecture here. We use a siamese CNN architecture for image verification. The architecture has two parallel convolutional sub-networks, which share their weights. The outputs of the two sub-networks are concatenated and fed into a single-stream fully-connected network. Our baseline architecture partially follows general building principles presented in the [22, 20], however, the main design goal was to make the architecture very simple. It

has three convolutional layers each followed by ReLU non-linearity and max-pooling that downsamples the data by the factor two. The convolutional layers are followed by one fully-connected and ReLU layer before the final classification layer. For a detailed overview, see Table 2.1.

Chapter 3

Data

In order to probe networks, we introduce a number of synthetic datasets. We keep these simple yet challenging for our task. Such a synthetic dataset can provide a baseline case in which only the factor of interests present a challenge. Since the task is image verification, the data consist of image pairs. Each pair contains either the same object or it represents two different objects. Furthermore, one of the images from a pair can be transformed. In this work we use two classes of transformations, 2D rotation and 2D translation.

We created several simple synthetic datasets consisting either of randomly generated polygon pairs or randomly generated pairs of doted patterns. Each pair in these datasets is transformed multiple times gradually increasing the amount of transformation. Furthermore, we also consider datasets that contain pairs that are rotated only by the same angle. This allows us to study effects of different rotations separately without affecting the training procedure by different rotations.

In this chapter we first provide an overview of dataset generation procedure and then we describe details and parameters of datasets used in the experiments.

3.1 Random polygons dataset

Generation of a synthetic dataset for a single rotation is described in detail by the Algorithm 1. The goal of this procedure is to generate random pairs of polygons. First we sample a list of polygon vertices. In order to create sibling polygon we randomly decide whether it is going to be the same polygon or a different one. In case of the different one, we sample another polygon. Since deciding whether two random polygons are the same or not would be an easy task as two randomly sampled polygons would be visually very different, we want to generate a polygon, which is similar to the first one. This is done by jittering the vertices of the original polygon as detailed in the Algorithm 3. We render the polygon with four times the required image size. This serves as a form of anti-aliasing and help us to reduce artifacts when transforming the image. Although we could transform the polygon vertices before rendering them, we want to keep the procedure consistent in case a dataset consisting of real images is used in the future.

When translation is used instead of rotation, the polygon sampling procedure samples a random polygon of the maximum relative size smaller than 1.0 to allow space for its

Algorithm 1: **Dataset generation.** m — number of polygons, α — rotation angle, s — image size, n_l, n_u — lower and upper bound for number of vertices, ϵ, σ — irregularity parameters, δ_l, δ_u — lower and upper bound for jittering of vertices, V — list of polygon vertices, I_i — image of the ith polygon, I'_i — image of the ith sibling polygon, SAMPLEPOLYGON — samples polygon as specified in the Algorithm 2, JITTERPOINTS — jitters polygon vertices as specified in the Algorithm 3, RENDERPOLYGON(V, s) — renders polygon of size s from the list of vertices V, IMROTATE(I_i, α) — rotates the image I_i by the angle α, IMRESIZE(I_i, a) — resizes the image I_i by the scale a

Require: $m, \alpha, s, n_l, n_u, \epsilon, \sigma$
for $i = 1, 2, \ldots, m$ **do**
  $V \leftarrow$ SAMPLEPOLYGON$(n_l, n_u, \epsilon, \sigma)$
  Sample $d \sim \mathcal{U}(0, 1)$
  if $d > 0.5$ **then**
    $V' \leftarrow V$
  else
    $V' \leftarrow$ JITTERPOINTS(δ_l, δ_u, V)
  end if
  $I_i \leftarrow$ RENDERPOLYGON$(V, 4s)$, $I'_i \leftarrow$ RENDERPOLYGON$(V', 4s)$
  $I'_i \leftarrow$ IMROTATE(I'_i, α)
  $I_i \leftarrow$ IMRESIZE$(I_i, 1/4)$, $I'_i \leftarrow$ IMRESIZE$(I'_i, 1/4)$
end for

translation within the canvas. The 10th (as used in the Algorithm 1) is replaced by a translation function. The translation function translates the image by a specified number of pixels t within a randomly chosen axis. The image is also translated in the remaining axis by a number of pixels sampled from $\mathcal{U}(0, d)$. The direction (positive or negative) in the both axes is also randomly sampled.

3.1.1 Generating random polygon

In order to generate random polygons, we use a method inspired by the [23]. The basic idea is to walk around a circle and take random angular steps. At each step we place a next polygon vertex at a random radius. Detailed explanation of this procedure procedure can be seen in the Algorithm 2. Although this formulation restricts the space of possible polygon shapes, it allows us to effectively control its irregularity.

3.2 Random dots dataset

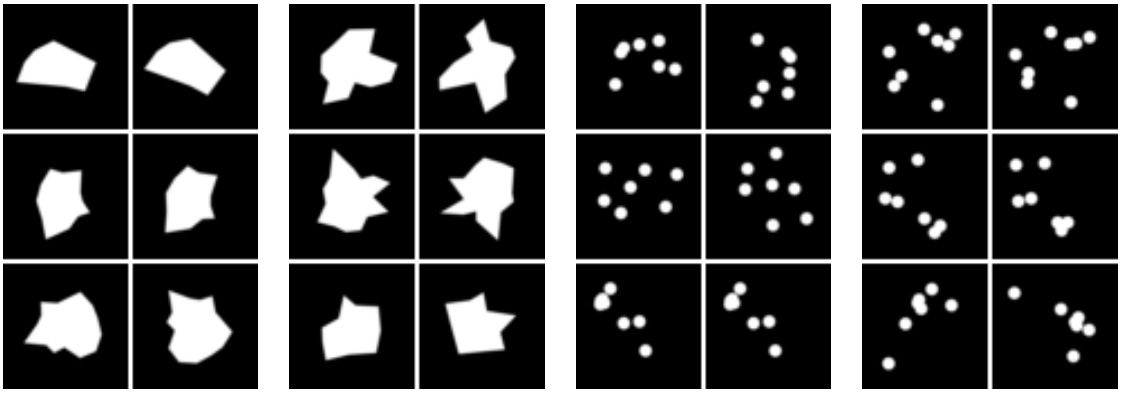
Generating the random dots dataset is very similar to the generation of the random polygon dataset. It only differs in two parts. First, instead of sampling vertices of a random polygon we uniformly sample centers of the dots on the image canvas. Second, instead of rendering polygon from the veritces we render dots at the sampled centers. The dots sampling procedure takes two arguments, the number of dots n in each image and their radius r.

Algorithm 2: **Random polygon sampling.** Samples a random polygon with vertices $(x_i, y_i) \in [-0.5, 0.5] \times [-0.5, 0.5]$. n_l, n_u — lower and upper bound for number of vertices, ϵ, σ — irregularity parameters

function SAMPLEPOLYGON($n_l, n_u, \epsilon, \sigma$)
  Sample $n \sim \mathcal{U}\{n_l, n_u\}$
  for $i = 1, 2, \ldots, n$ **do**
    Sample $\delta_i \sim \mathcal{U}(\frac{\pi}{n} - \epsilon, \frac{\pi}{n} + \epsilon)$ and $r_i' \sim \mathcal{N}(0.25, \sigma^2)$
    $\theta_i \leftarrow \theta_{i-1} + \frac{1}{k}\delta_i$, $k = \frac{\sum_{i=1}^{n} \delta_i}{2\pi}$
    $r_i \leftarrow clip(r_i', 0, 0.5)$
    $x_i \leftarrow r_i \cos(\theta)$, $y_i \leftarrow r_i \sin(\theta)$
  end for
  return list of points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
end function

Algorithm 3: **Points jittering.** δ_l, δ_u — lower and upper bound for jittering of points, $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ — list of points

function JITTERPOINTS(δ_l, δ_u, $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$)
  for $i = 1, 2, \ldots, n$ **do**
    Sample $\delta_x \sim \mathcal{U}(\delta_l, \delta_u), \delta_y \sim \mathcal{U}(\delta_l, \delta_u)$, $s_x \sim \mathcal{U}(-1, 1)$, and $s_y \sim \mathcal{U}(-1, 1)$
    $x_i \leftarrow x_i + sign(s_x)\delta_x$, $y_i \leftarrow y_i + sign(s_y)\delta_y$,
  end for
  return list of points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
end function



(a) (b) (c) (d)

Figure 3.1: **Datasets examples.** (a) and (b) contains random samples from the poly-m dataset, (b) and (c) from the dots-m dataset. (a) and (c) show pairs form the "same objects" class, (b) and (d) pairs from the "different objects" class.

When translation is used, the same modifications as presented in subsection 3.3.2 apply also here. The dots are not sampled on the whole canvas, but a margin of size of the maximum applied translation is used along the canvas edges.

3.3 Datasets details

In this section, we provide detailed description of dataset used in our experiments. Our design goal was to make the dataset reasonably challenging for the proposed architectures.

3.3.1 Datasets using rotation

The dataset with multiple rotations has 10000 samples in training set, 1000 in validation set and 5000 in testing set for each rotation. We call this variant *poly-m*. The datasets with a single rotations have 20000 samples in the training set. We had to double the number of training images per rotation in comparison with the *poly-m* dataset, as the network did not have enough training samples and was subject to extreme overfitting. The rest is the same, i.e. 1000 in the validation set and 5000 in the testing set. We call them *poly-s-*α, where α is the applied rotation in degrees. We use discrete rotations $\alpha \in 0, 15, \ldots, 180$ degrees. The same set of polygon pairs is used for all the rotations. All the datasets have grayscale images of the size 56×56 and were generated with the parameters specified in the Table 3.1.

Table 3.1: **Datasets parameters.** The parameters naming is the same as in the Algorithm 1

parameter	*dataset*	n_l	n_u	ϵ	σ	δ_l	δ_u
value	*polygons*	6	16	0.5	0.26	0.2	0.3
	dots	6	8	–	–	0.2	0.3

The random dots datasets, called *dots-s-α* and *dots-m*, have the same parameters as the random polygons datasets specified in the subsection 3.3.2 unless stated otherwise. The radius r to the 5% of the image size. For further details, see the Table 3.1.

3.3.2 Datasets using translation

Datasets using translation are called *poly-tran-m* and *dots-tran-m*. They have the same parameters as poly-m and dots-m respectively except for the following changes. The maximum relative size of the polygon is 0.7, the canvas margin for the dots is 15% of the image size (image width or height). We used 13 different values, the same number used for the rotation dataset, for the translation size t evenly spaced between 0% and 15% of the image size. We choose not to use bigger translations, as it would require to sample smaller polygons or dot patterns. Given the small image size, their shape would be significantly distorted and thus very difficult to classify. On the other hand, using bigger images would require to design a different and more complex network architecture than the one we intended to use.

Chapter 4

Experiments

In this chapter, we firstly investigate the dependence of classification error on the rotation between two input images. Secondly, we discuss two ways of implementing mental rotation and examine whether it can improve verification accuracy.

Unless otherwise stated, we use the following hyper-parameters for all the experiments. We use standard stochastic gradient descent with momentum. We set the batch size to 100 and learning rate to 0.001. The learning rate is decreased by the factor 10 after every 150th epochs. We train for 450 epochs. The momentum is 0.9 and the weight decay is 0.0005. We run each training 4 times with a different random seed and report the average. We use MatConvNet toolbox [24] for the implementations of the experiments.

4.1 Effect of transformations on verification performance

We use the network architecture as presented in the section 2.2 in the following experiments. Since the task is object verification, we classify the input pairs into two classes — both images contain the same object, images contain two different objects. Thus, the last fully-connected layer has two units.

First, we train networks on the poly-m and dots-m datasets. Throughout this work, we name the networks after the datasets they were trained on. The Figure 4.1 shows test errors with respect to the rotation between the two inputs. The lowest test error is on the zero rotation subset. The test error sharply increases when the images are rotated by 15 degrees. Contrary to our expectation, the test error does not globally increase with bigger rotations. Furthermore, it has another two local minima in 90 and 180 degrees. The character of the test error is similar for the both datasets.

As the results oppose our prior belief, we trained individual networks for each rotation using the datasets *poly-s-*α and *dots-s-*α. This can be regarded as a form of sanity check. As can be seen form the Figure 4.1, although test error is lower the overall character with the same minima at the 0, 90, 180 degrees is similar to the one from the previous experiment.

A possible explanation for the minima at 90 degrees and 180 degrees can be that the performance suffers from quantization effect. When not rotating by a multiplier of 90 degrees, the pixels of the rotated image will not lay on a discrete coordinates and the resulting image has to be interpolated. This can lead to loss of information about subtitles

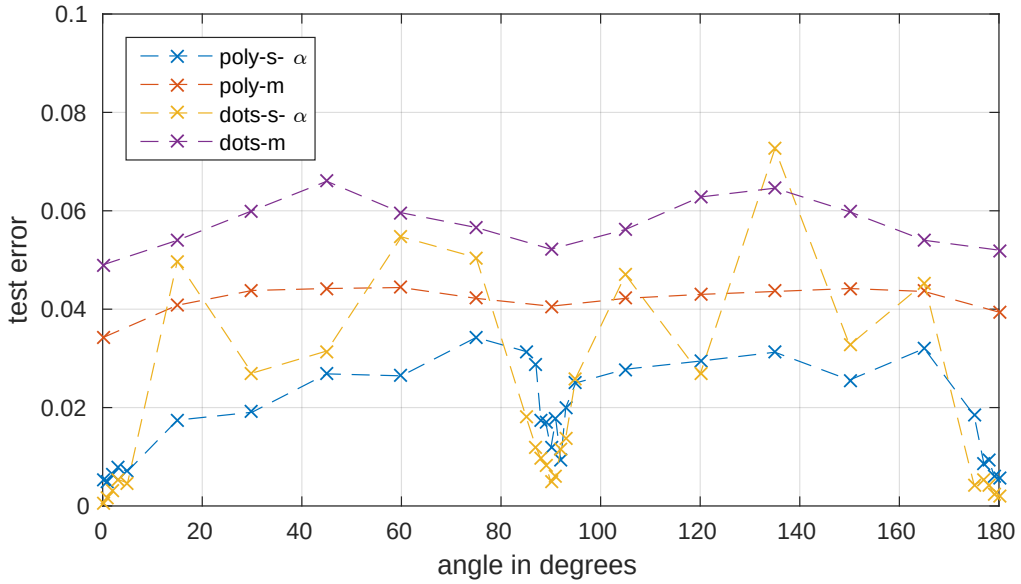


Figure 4.1: **Test error with respect to the rotation.** The figure shows test errors on subsets of the particular datasets. Notice that all the networks have local minima in around 0, 90, and 180 degrees. Therefore we evaluated additional angles around them for the poly-s-α and dots-s-α datasets.

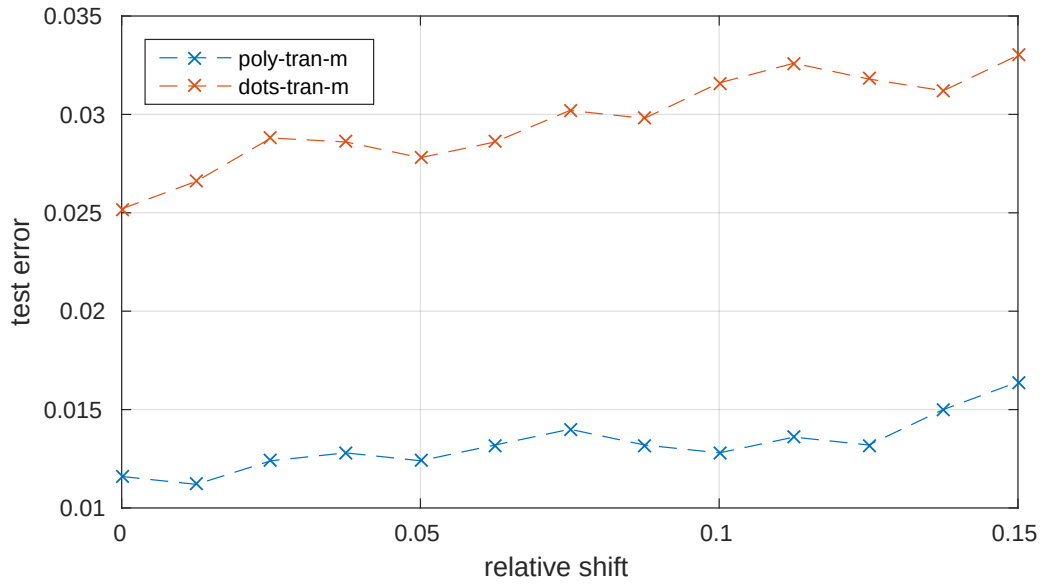


Figure 4.2: **Test error with respect to the translation.** The figure shows test errors on subsets of the particular datasets. The values on the x axis of this figure correspond to the maximum absolute horizontal or vertical shift. The shift is relative to the image size. Notice the growing trend of error with respect to the absolute shift.

in object geometry needed for successful recognition. Furthermore, consider a case when the CNN learns to detect for example a part of polygon using its convolutional layers. (We can expect this as CNNs are known to have strong responses for discriminative object parts in their convolutional layers [25].) If the CNN wants to detect the same polygon part in the rotated image, it can learn the rotated version of the filters detecting this polygon part. However, all the rotations except for the multipliers of 90 degrees would require to interpolate the rotated filter. This can again negatively effect the performance, since the interpolated filter can respond to slightly different visual stimuli.

Overall, both the poly-m and dots-m networks perform well with test error 4.2% and 5.7% respectively. The differences in performance with respect to rotation are relatively small, at most 23.0% and 25.8% respectively of relative difference. For comparison, we trained the same network on the translation datasets. The test error tends to increase with the size of translation as can be seen in Figure 4.2. The overall test error is 1.32% for the poly-tran-m dataset and 2.97% for the dots-tran-m dataset. The network performs better on translations. This can be also partly caused by the small translations we used.

4.2 Mental rotation

The previous experiments with the lowest error at 0 degree rotation suggest that aligning objects in the input images can improve performance. In this section we aim to built such a mechanism directly into the network. We first start with a simple bruteforce method and continue with more sophisticated mechanism. All the experiments are using verification network trained on only 0 rotation subset of poly-m dataset.

4.2.1 Exhaustive mental rotation

The idea of the exhaustive mental rotation is very simple. We train a neural network which is specialized to verify only aligned polygons. In the test time, we exhaustively try all the possible rotations, in our case the translations are discretized by 15 degrees. We classify an input pair of objects as the same if and only if it has been classified as such at least for one rotation. The input pair is otherwise classified as two different objects. This greatly reduces the test error to 1.73% om the poly-m dataset and 0.57% on the dots-m dataset. However, it also multiplies the evaluation time by the number of tested rotations if not run in parallel.

4.2.2 Spatial transformer

Encouraged by the exhaustive mental rotation method results, we wanted to take advantage of the spatial transformer module, which would correctly align the inputs in one pass before feeding them to classification network. This would have clearly positive impact on the evaluation speed and possibly also on accuracy, since it should avoid classifying incorrectly aligned images.

4.2.2.1 Rotation prediction

However, as the spatial transformer requires localization network predicting transformation parameters, we first verify whether our simple architecture has ability to estimate the angle of rotation between the two input images from our datasets. We use the same architecture as in the verification experiments, but we replace the classification layer with a layer that classifies the input pair into classes from the set $\{0, 15, \ldots, 180\}$. We used the poly-m dataset for training and call this neural network poly-cls. The network achieves very low testing error, only 2.11%.

4.2.2.2 Combining networks

The networks poly-cls and poly-m can be combined together. The first network would predict transformation parameters, in our case angle of rotation. This would by used by the spatial transformer module, which would transform one of the input images, before it is along with the second image fed into the poly-m verification network.

This fusion can operate in two ways. First, we can directly use the pretrained poly-cls localization network and the pretrained classification network in test mode. Second, we can train the combined networks end-to-end from scratch and the localization network should learn to predict transformations implicitly (in unsupervised manner). This can be expected as it was observed in the original work [10].

4.2.2.3 Explicit rotation

When trained explicitly to transform the input image, the network works exceptionally well with testing error 0.64% on the poly-m dataset and 0.40% on the dots-m dataset outperforming the basic poly-m and dots-m architectures and the brute force method. It is also considerably faster than the latter method, the test time is only 2 times higher when compared to the poly-m network. This is a bit unfair comparison as such a network has two fold more learnable parameters that the original net. To make the comparison fair, we modified the architectures. The localization network has then 70% of parameters from the original network and the classification network has 30%. The classification net is highly negatively affected by such a drastic reduction of parameters, however, we were not able to lower the number of parameters in the localization network even further, since it was not possible to optimize the network at all when using less parameters. This combination of networks still has lower test error, 3.83%, than the original poly-m architecture.

4.2.2.4 Implicit rotation

However, when we were trying to replicate results from the [10] on their rotated version of the MNIST dataset, we found out that in many situations the spatial transformer network does not learn to transform the input images. We had to experiment with various architecture modifications to make the spatial transformer rotate the input into its canonical proposition. We did not experience such a difficulties in case of the translated MNIST dataset. We still do not quite understand what kind of conditions make the spatial transformer learn to transform the inputs. One of our hypothesis is that the architecture is very sensitive to number of

weights in the localization and classification networks. If the number of parameters is high in the classification network, then it can easily overfit on the training data and before the localization network learns to predict the correct rotation, which is probably more difficult to predict than translation is. This also happened when we were trying to adapt spatial transformer on this task as we were not able to find such an architecture that would do the transformation.

Chapter 5

Features analysis

The experiments from the chapter 4 showed that there is relatively small difference in testing error between different rotations and translations. This means that the network has to learn, at least to some extent, rotation and translation invariant features while preserving information about the geometry in order to successfully verify the images. We want to probe the learned features in both the convolutional layers and in the fully connected layers by inverting them back to the image space.

Recently, there has been two methods introduced for this purpose. The work by Mahendran & Vedaldi [15] uses gradient descent optimization in order to find a regularized image that has a similar representation to the original image representation in feature space. Formally, let $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ be the input image and $\phi(\mathbf{x}) \in \mathbb{R}^d$ its representation in feature space. The goal is to find the image $\mathbf{x}^* \in \mathbb{R}^{H \times W \times C}$ that minimizes the objective

$$||\phi(\mathbf{x}) - \phi(\mathbf{x}^*)||_2^2 + \lambda \mathcal{R}(\mathbf{x})$$

where $\mathcal{R} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}$ is a regularizer and $\lambda \in \mathbb{R}_+$ regularization trade-off parameter. The resulting image $\mathbf{x}^*$ can be than thought of as the inversion of the representation $\phi(\mathbf{x})$ or the reconstruction of the original image $\mathbf{x}$ from its features $\phi(\mathbf{x})$.

Dosovitskiy & Brox [5] train up-convolutional network to perform the image reconstruction. Let $\phi'(\phi(\mathbf{x}); \mathbf{w}) : \mathbb{R}^d \to \mathbb{R}^{H \times W \times C}$ be a neural network that takes the image representation $\phi(\mathbf{x})$ outputs the image reconstruction $\mathbf{x}^*$. The training procedure finds such a weights $\mathbf{w}$ that minimize the objective

$$\sum_i ||\mathbf{x}_i - \phi'(\phi(\mathbf{x}_i); \mathbf{w})||_2^2$$

where $\mathbf{x}_i \in \mathbb{R}^{H \times W \times C}$ are training images. This method produces better image reconstructions by automatically learning image prior, which is then encoded in the network weights. However, this is undesirable for our work as we want to visualize only the information encoded in the features. Therefore, we choose the first method.

In practice, we used gradient descent with momentum and a simple L2 norm as the regularizer $\mathcal{R}(\mathbf{x}) = ||\mathbf{x}||_2^2$. We set the learning rate to 0.1, momentum to 0.9. If not stated otherwise, we use reguralization trade-off parameter λ set to 0.0005.

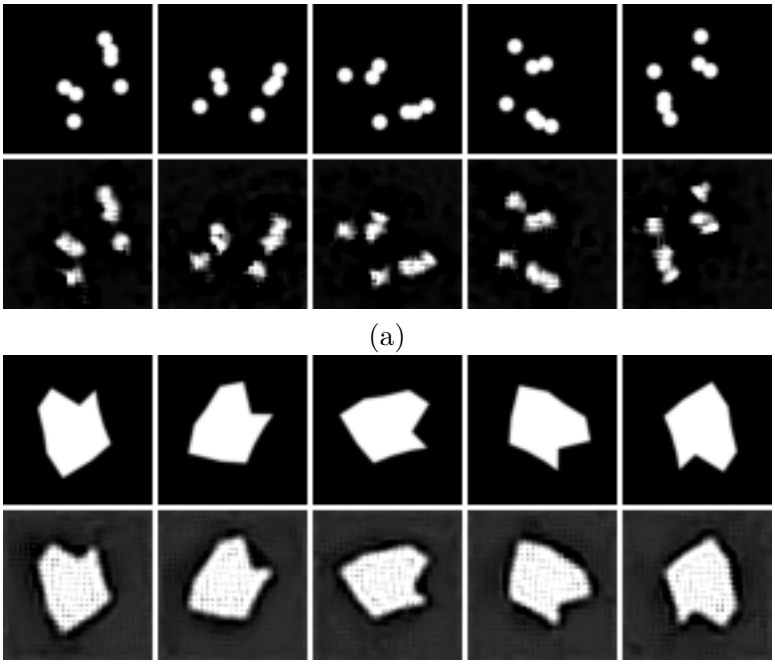
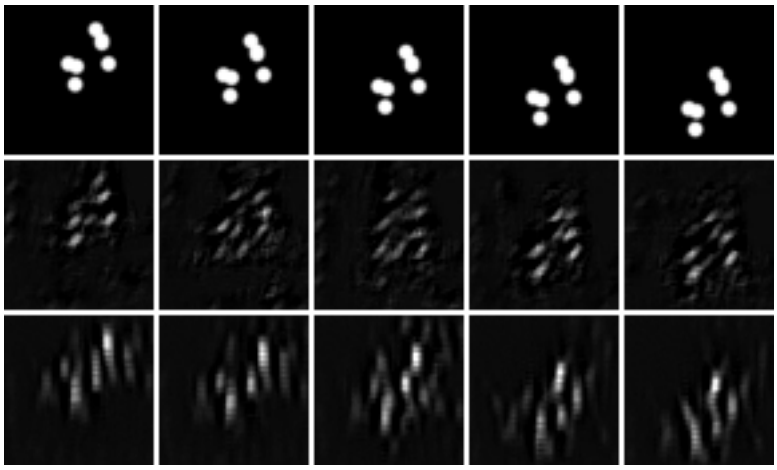


Figure 5.1: **Convolutional features reconstruction: rotation.** The first rows of (a) and (b) contain samples from the dots-m and poly-m datasets and their consecutive rotations by 45 degrees. The second rows show their corresponding reconstructions from the last pooling layer.

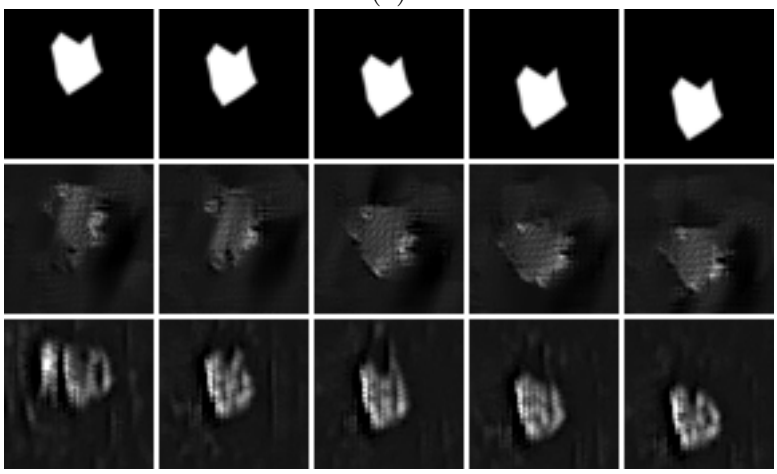
Note that in case of reconstruction from the fully-connected layers we have to reconstruct both of the images from the input pair as our siamese architecture is already merged into a single stream network starting from the first fully-connected layer.

5.1 Convolutional features

The convolutional part of a CNN has the most transformation invariant features in its last layer [13]. Therefore, we invert the features of the last pooling layer just before the two streams of the network join. The reconstructions Figure 5.1 from the poly-m network show that the features preserves information about object geometry and its location. The reconstructions of the of the polygons have a little bit diminished bodies as only edges are intuitively needed for successful recognition. It is clear that the features are not rotation invariant and we have to look deeper into the network. This is with contrast with features from the networks trained on translated data. The Figure 5.2 suggests that the features are more localy invariant to transformation, in this case translation.



(a)



(b)

Figure 5.2: **Convolutional features reconstruction: translation.** The first row in (a) and (b) contain samples from the dots-m and poly-m datasets and their consecutive translations. For illustrative purposes, we use only vertical translation. The second and third rows show their corresponding reconstructions from the last pooling layer. The network used for the second row was trained on the standard dataset as described in the 3rd chapter. Notice that the reconstructions in (a) contain diagonal strips even though the translations were evenly distributed to all directions. The third row was trained on a modified dataset where only vertical translations were allowed. That corresponds to the directions of the strips in (a). Notice that the reconstructions are noticeably less accurate than those from Figure 5.1.

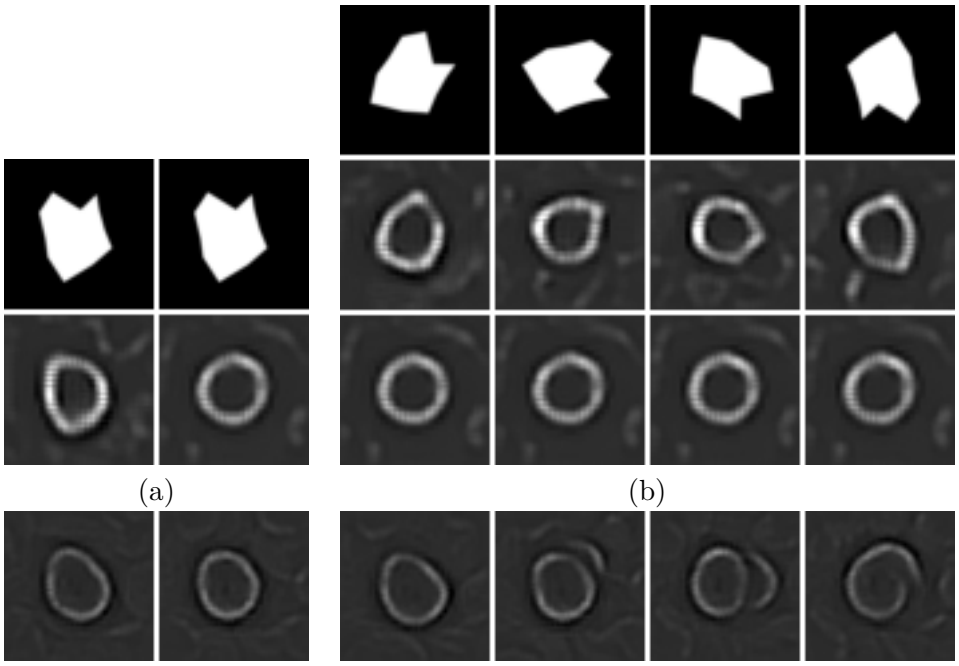
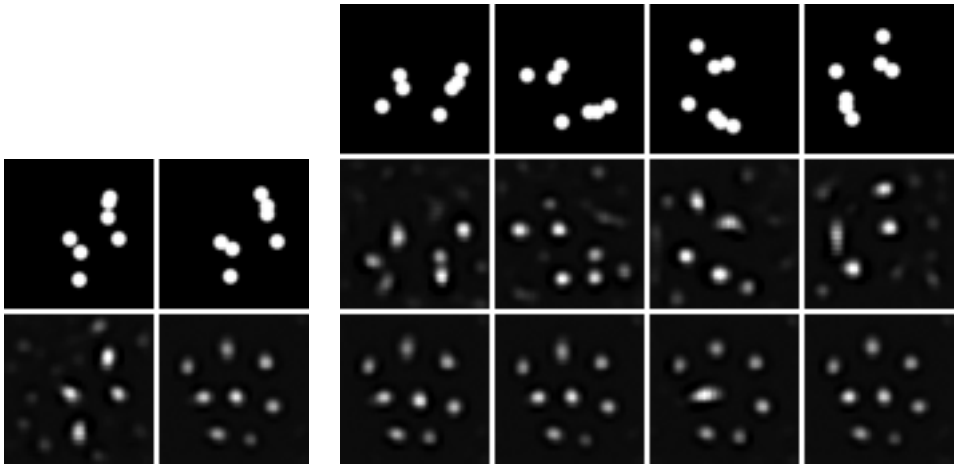


Figure 5.3: **Reconstructions from the first FC layer: poly-m dataset.** The first row in (a) contains a pair of the same input images from the dataset poly-m. The second row contains their corresponding reconstructions. The first row in (b) contains rotated versions of either the first input or the second as described further. The second row in (b) contains reconstructions of the first image from the input pair. In this case, the first image was consecutively rotated by 45 degrees while the second remained the same. Notice that as the input image in the first row is rotating its reconstructions from the second row seems to be rotating as well. The third row in (b) contains reconstructions of the second input. In contrast with the previous case, the second input was rotated whereas the first remained the same. Notice, that the reconstructions do not significantly change with respect to the rotation. These phenomenas are discussed further in the section 5.2. (c) contains reconstructions of the input pair from the poly-m-shfl network and (d) reconstructions of the rotated input. We do not observer any of the behaviors that were presented in (b). The reconstructions of the second input (not shown here) behave similarly. Note that in (b) we show only reconstructions of one of the inputs, as the reconstructions of the other one remain almost the same.



(a) (b)

Figure 5.4: **Reconstructions from the first FC layer: dots-m dataset.** This figure displays data obtained from the dataset dots-m. See Figure 5.3 for detailed annotation, as it applies also here. Notice that the reconstructed dots from the first input roughly corresponds to those in the original input, whereas the reconstructed dots from the second input make completely different pattern.

5.2 Fully-connected features

The reconstructions from the fully-connected features provides us with several interesting insights. The Figure 5.3 contains reconstructions for the polygons dataset. Reconstructions for the dots dataset are included in the Figure 5.4. The reconstructions of both inputs now contain only the edges of the polygons. The reconstructions of the first input are quite distorted, but still partly retains the original shape. The reconstructions of the second image from the input pairs are not visually similar to the original images and, quite interestingly, they almost do not change when the input image is rotated. This suggests that the second input images are mapped onto rotation invariant features. In contrast, when the first input images are rotated the reconstructions seems to be rotated as well. The reconstructions of the dots are subject to the same phenomena.

In order to quantitatively confirm this, we measure Euclidean distance between the zero rotation features and inputs with bigger rotations. The Figure 5.5 shows that when the second input is rotated the Euclidean distance is smaller than when the first input is rotated. It seems that the network has adapted to a small bias presented in the dataset. The dataset consists of pairs of polygons. When the rotated version is created, only the second images from the pairs are rotated. If the images in the pairs are shuffled in the dataset, this phenomena disappears. We call the network trained on this modified dataset *poly-m-shfl*. Figure 5.5 also shows the Euclidean distances of the features from the poly-m-shfl are almost the same regardless of the input rotated, as the network has adapted to rotations of the both inputs.

However, the reconstructions from poly-tran-m and dots-tran-m networks are not subject to the same phenomena as poly-m and dots-m nets. The reconstructions move together with

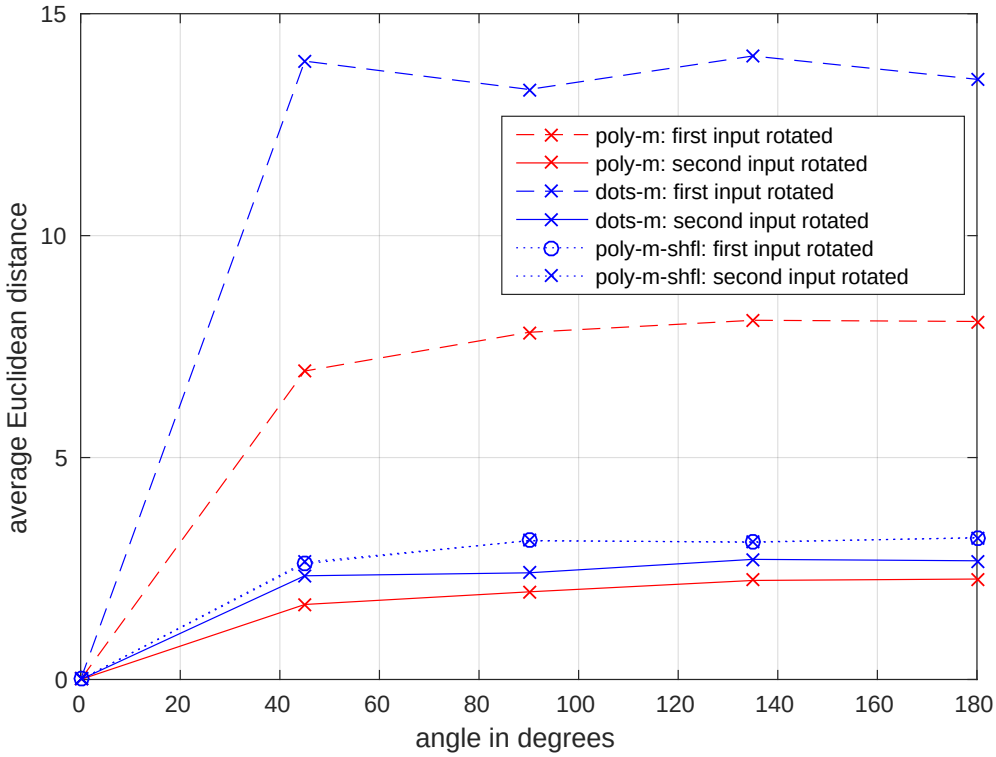
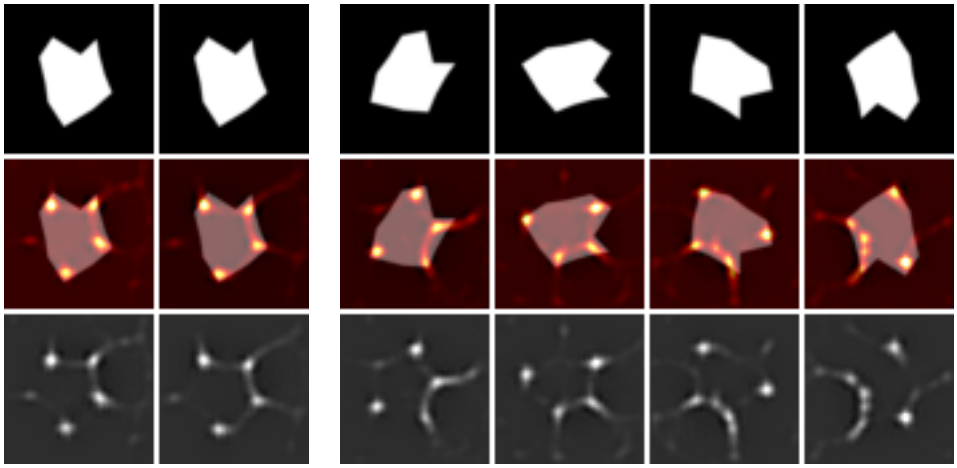


Figure 5.5: **Average Euclidean distances between fully-connected features with respect to the rotation.** The figure shows the Euclidean distances between the fully-connected features of the aligned input images and their rotated versions. Notice that when the first input is rotated the Euclidean distances are significantly higher than if the second input is rotated. However, this is not the case of poly-m-shfl network, as it performs almost equally well in the both cases. This is discussed further in the section 5.2. The distances for the poly-m network are smaller when compared to those of the dots-m network. This can be possible reason why the verification accuracy of the latter network is worse when compared to the former network.



(a) (b)

Figure 5.6: **Reconstructions from the FC features of the poly-cls network.** The first row corresponds to the inputs, the third row their reconstructions and the second row overlays the first row with the hot color-mapped version of the reconstructions. (a) shows the whole input pair. (b) shows only the rotated input. Notice that the brightest regions in the reconstructed images correspond to the polygon vertices.

their original images. The figures are enclosed in the appendix.

Furthermore, we reconstruct the inputs for the rotation classification network poly-cls. As can be seen from the Figure 5.6 the features represents only the most distinctive parts of polygons, their vertices. The figure furthermore suggests that the network has implicitly learned to find correspondences of between the two inputs in order to be able to correctly estimate rotation between the two inputs.

We observed that using the regularization, i.e. using λ higher than zero, negatively affects the quality of reconstructions. We measure the quality as the classification error on the testing dataset that has the original images replaced by their reconstructions. When using $\lambda = 0.0005$, the classification error for the poly-m dataset is 8.3%, 9.1% for the dots-m dataset, and 3.0% for the poly-cls dataset. Without the regularization it is 5.9%, 6.7%, and 2.4% in the same order. This is much closer to the original error. However, the regularization is very useful in our situation, as it encourages reconstruction of only the most significant parts of the input image from the viewpoint of the particular representation. We provide some unregularized reconstructions in the Appendix A.

Chapter 6

Conclusion

The first goal of this work was to probe how the classification error of deep convolutional neural networks used for image verification depends on transformation between two visually-similar images. Secondly, we wanted to investigate whether the performance of such networks can be improved by "mentally rotating" the input images before they are fed in to the verification network. In order to do so, we considered 2D rotations and 2D translations and we created several synthetic datasets consisting of randomly sampled polygons and dots patterns. Generating synthetic data allowed us to fully control the properties of the datasets used in the experiments. We showed that networks achieve their optimal performance when, as it could be expected, no rotation is presented. Other local minima are in the close neighbourhood of 90 and 180 degrees. However, performance is surprisingly uniform, and local minima in the verification error are only slightly higher than the global minimum. In case of translation, the behaviour is similar and the error tends to increase with increasing the translation. Both results hold for the two types of objects, polygons and dotted patterns.

Continuing with the investigation of "mental rotations", we first tested a method that exhaustively applies all possible rotations to one of the inputs to verify it against the other. This simple approach leads to a substantial improvement in verification accuracy, but has the disadvantage of being computationally intensive. In order to solve this problem, we tested the idea of using a spatial transformer subnetwork to predict in one go the correct *relative rotation* to apply to the inputs (to the best of our knowledge, this is the first time spatial transformers are applied to align image pairs). The spatial transformer network performed very well on this problem if image alignment and was therefore incorporated in verification by means of a spatial resampler layer. In this manner, we were able to further lower the verification error compared to the brute force approach, with a significantly smaller number of operations overall. However, as we have discussed in detail in the experiments, we were not able to train this combination end-to-end from scratch.

The results in the previous experiments showed the remarkable performance of deep architectures in handling transformations. This led us to investigating further the learned features. We used a feature inversion method to visualize the features of the last pooling layer, which can be expected to be the most transformation invariant layer in the convolution part of the network, and the first fully connected layer (FC1). We discovered rotation invariant properties of the first fully connected layer, which we also confirmed quantitatively by measuring Euclidean distances between features of differently rotated inputs. We

did not observe such a phenomena for the translation transformation. Furthermore, our results also suggest another intriguing properties of the FC1 features. When we trained the CNN to predicted rotation between two inputs, the FC1 features implicitly learn to match correspondences between the rotated inputs.

Our study of transformation properties of CNNs has revealed several interesting behaviours of networks trained to verify pairs of transformed images. However, our understanding of the underlying statistical mechanisms, i.e. how such properties emerge in networks, remain limited, and should be targeted by future explorations.

Bibliography

[1] Mathieu Aubry and Bryan C Russell. Understanding deep features with computer-generated imagery. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2875–2883, 2015.

[2] Taco S Cohen and Max Welling. Transformation properties of learned visual representations. *arXiv preprint arXiv:1412.7659*, 2014.

[3] Lynn A Cooper. Mental rotation of random two-dimensional shapes. *Cognitive psychology*, 7(1):20–43, 1975.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[5] Alexey Dosovitskiy and Thomas Brox. Inverting visual representations with convolutional networks. *arXiv preprint arXiv:1506.02753*, 2015.

[6] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[7] François Fleuret, Ting Li, Charles Dubout, Emma K Wampler, Steven Yantis, and Donald Geman. Comparing machines and humans on a visual categorization test. *Proceedings of the National Academy of Sciences*, 108(43):17621–17625, 2011.

[8] Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15(6):455–469, 1982.

[9] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[10] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2008–2016, 2015.

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

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] 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.

[14] 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.

[15] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[16] Roland Memisevic. Gradient-based learning of higher-order image features. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 1591–1598. IEEE, 2011.

[17] Roland Memisevic. Learning to relate images. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1829–1846, 2013.

[18] Roland Memisevic and Geoffrey E Hinton. Learning to represent spatial transformations with factored higher-order boltzmann machines. *Neural Computation*, 22(6):1473–1492, 2010.

[19] RN Shepard and J Metzler. Mental rotation of three-dimensional objects. *Science (New York, NY)*, 171(3972):701, 1971.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[22] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

[23] Mike Ounsworth (<`http://stackoverflow.com/users/1907046/mike-ounsworth`>). Algorithm to generate random 2d polygon. Stack Overflow (<`http://stackoverflow.com/users/1907046/mike-ounsworth`>). [Online; accessed 22-April-2016].

[24] Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 689–692. ACM, 2015.

[25] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer vision–ECCV 2014*, pages 818–833. Springer, 2014.

[26] Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip HS Torr. Conditional random fields as recurrent neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1529–1537, 2015.

Appendix A

Additional reconstruction visualizations

Here we provide additional reconstructions from the first FC layer. In each figure, title specifies the dataset the network was trained on. The rows in the figures corresponds to a single input pair and its corresponding reconstructions.

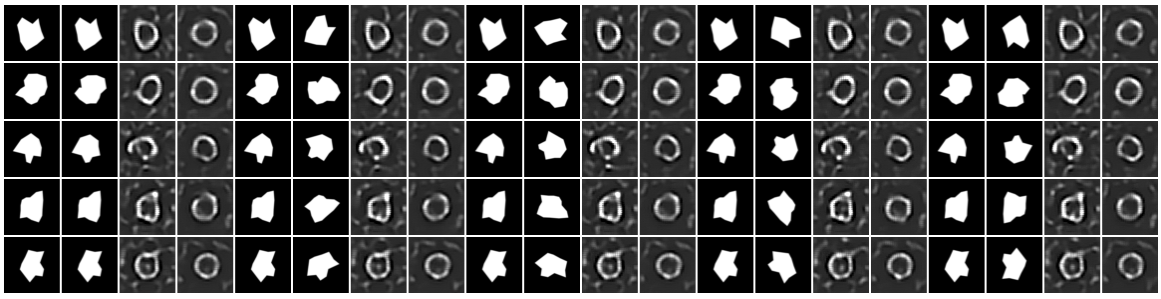


Figure A.1: **poly-m (a)**

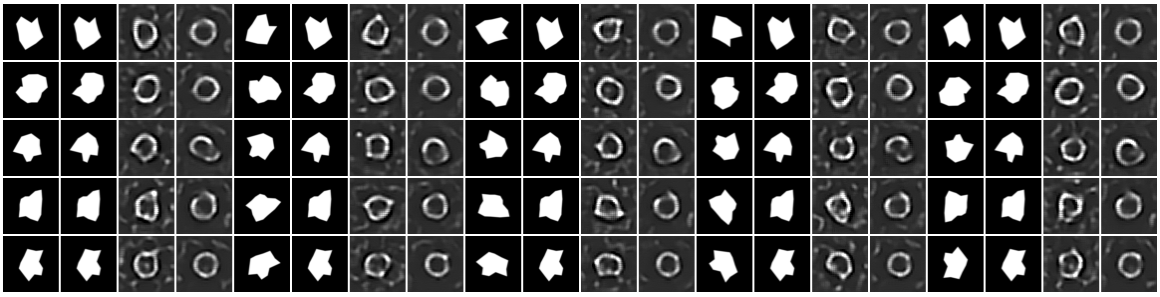


Figure A.2: **poly-m (b)**

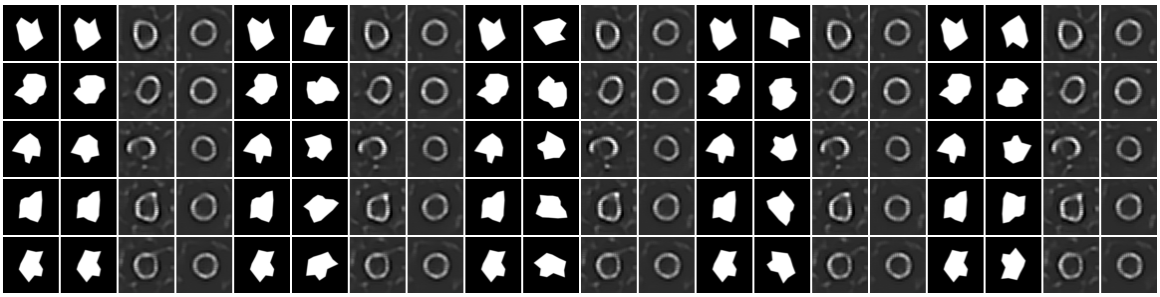


Figure A.3: **poly-m regularized (a)**

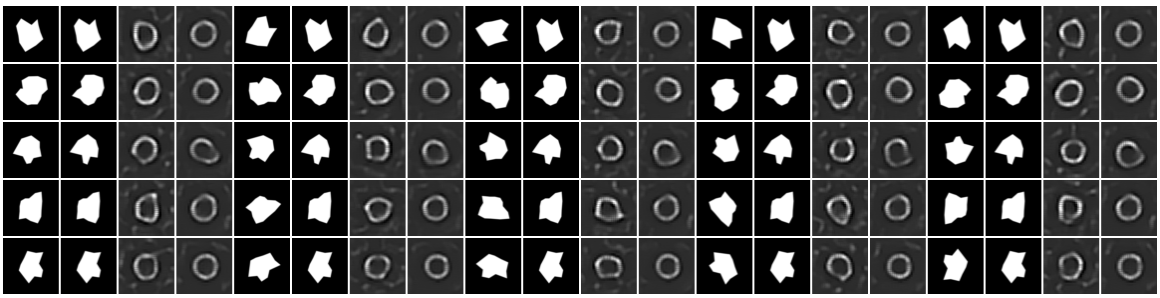


Figure A.4: **poly-m regularized (b)**

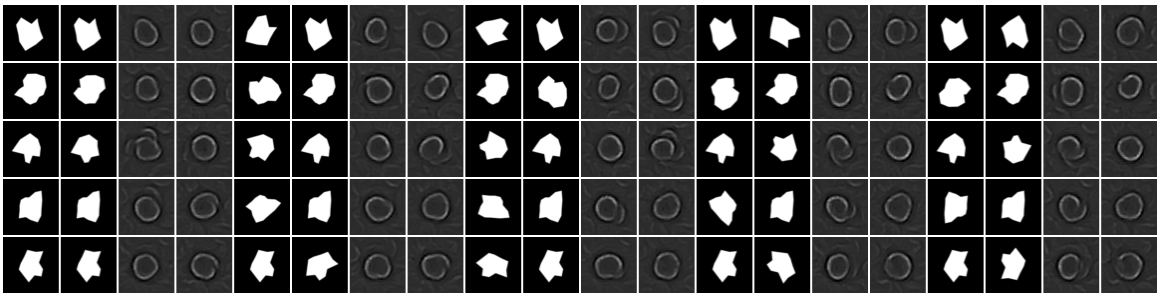


Figure A.5: **poly-m-shfl regularized**

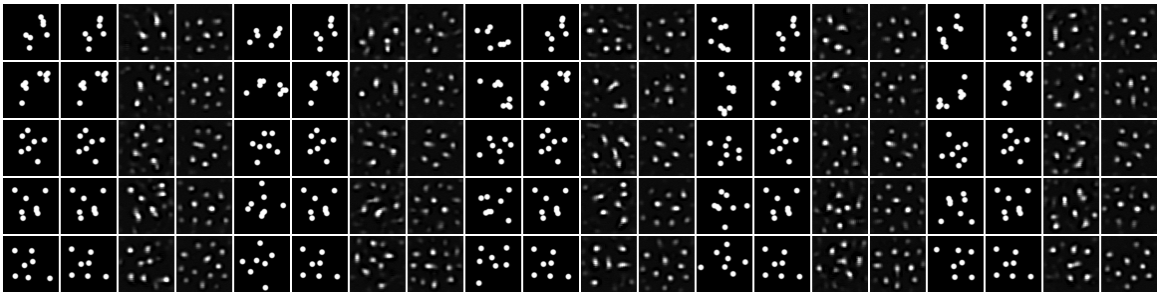


Figure A.6: **dots-m**

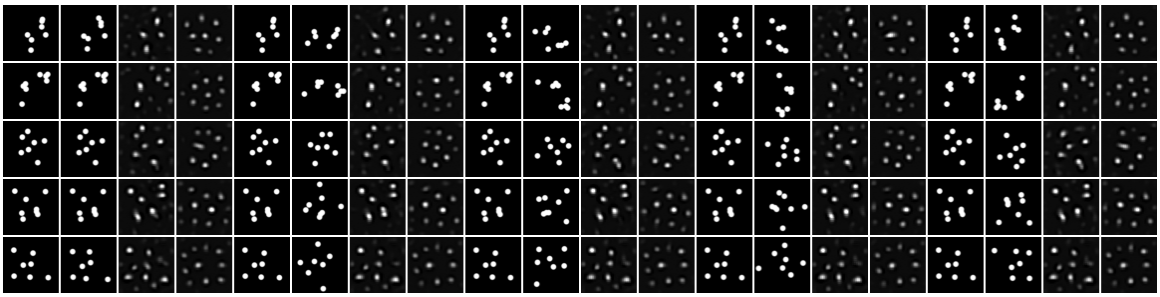


Figure A.7: **dots-m regularized (a)**

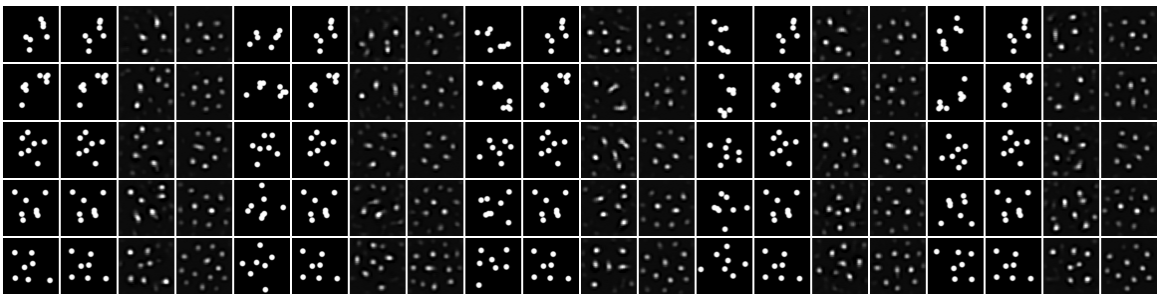


Figure A.8: **dots-m regularized (b)**

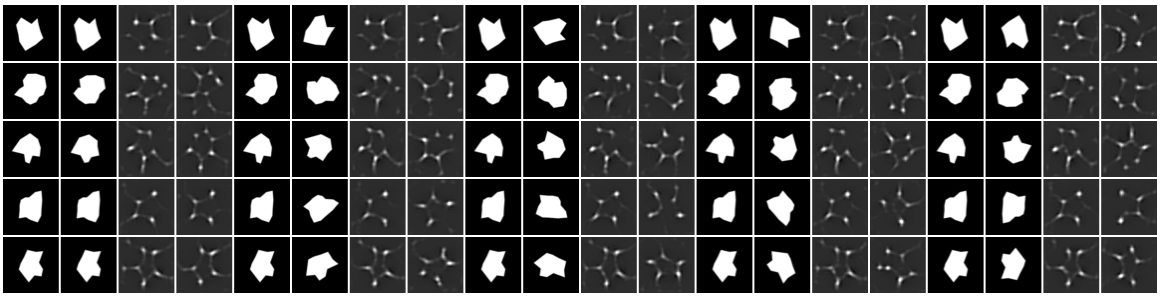


Figure A.9: **poly-cls regularized**

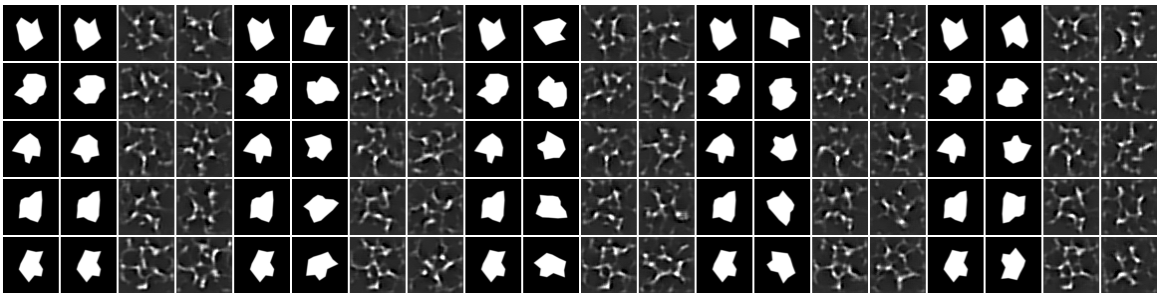


Figure A.10: **poly-cls**

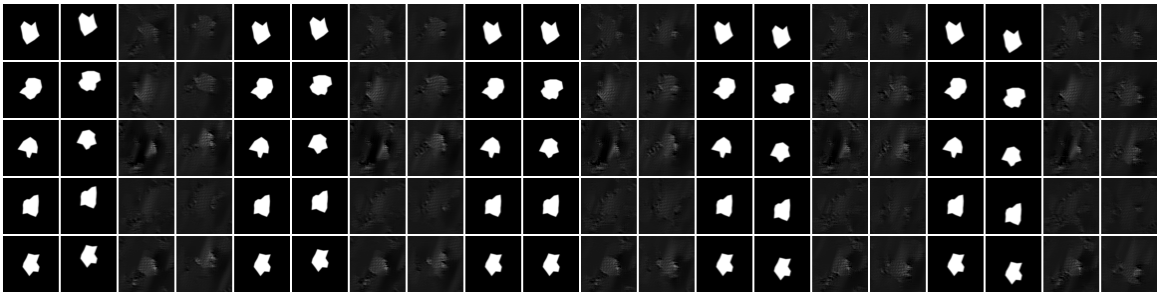


Figure A.11: **poly-tran-m**

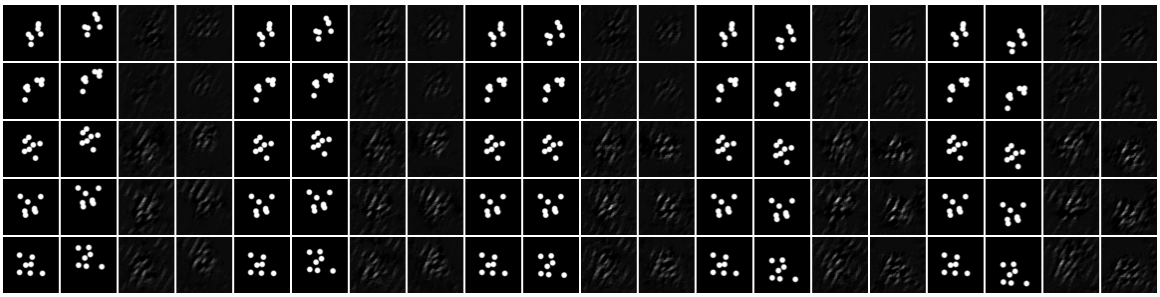


Figure A.12: **dots-tran-m**

Appendix B

Nomenclature

2D	two-dimensional
CNN	convolutional neural networks
FC	fully-connected
FC	fully-connected
RBM	Restricted Boltzmann Machine
ReLU	rectified linear unit