Deep Diffeomorphic Transformer Networks

Nicki Skafte Detlefsen
Technical University of Denmark
nsde@dtu.dk

Oren Freifeld
Ben-Gurion University
orenfr@cs.bgu.ac.il

Søren Hauberg
Technical University of Denmark
sohau@dtu.dk

Abstract

Spatial Transformer layers allow neural networks, at least in principle, to be invariant to large spatial transformations in image data. The model has, however, seen limited uptake as most practical implementations support only transformations that are too restricted, e.g. *affine or homographic maps, and/or destructive maps, such as thin plate splines. We investigate the use of flexible diffeomorphic image transformations within such networks and demonstrate that significant performance gains can be attained over currently-used models. The learned transformations are found to be both simple and intuitive, thereby providing insights into individual problem domains. With the proposed framework, a standard convolutional neural network matches state-of-the-art results on face verification with only two extra lines of simple TensorFlow code.*

1. Introduction

Models that are invariant to spatial transformations of the input are essential when designing accurate and robust image classifiers; *e.g.*, we want models that can separate objects' *shape* and *appearance* from their *position* and *pose*. *Convolutional Neural Networks (CNNs)* [33] achieve some invariance and produce state-of-the-art results in, *e.g.*, classification [26, 44, 52], localization [47, 55], and segmentation [36, 43]. This is partially achieved by the translation invariance of the convolutional layers and partially by the (local) spatial invariance in the max-pooling layers.

Current CNN architectures, however, typically employ small pooling regions (*e.g.*, 2×2 or 3×3 pixels), thereby limiting their invariance to large transformations of the input data. To counter this, Jaderberg *et al*. [27] introduced the *Spatial Transformer (ST)* layer which explicitly allows for spatial manipulation of data within the network. Via a regression network, the ST-layer learns an input-dependent transformation such that together with the subsequent layers the overall network achieves optimal performance. Models with an ST-layer have either increased classification accuracy [25, 42] or maintained the same performance level

Figure 1: The spatial transformer layer improves performance of deep neural networks for face verification. By learning an affine transformation, the network can "zoom in" on the subjects face; when learning a flexible transformation (proposed), the network here stretches an oval face to become square. This significantly improves performance.

with a simpler network [11]. We argue that the potential of the ST-layer has yet to be fully utilized. Particularly, while in theory the ST-layer allows for any parametrized family of differentiable transformations, only certain types of maps – affine, homographies, or thin plate splines (TPS) – appear in most practical implementations [25, 35, 42]. We note that affine maps and homographies are of limited expressiveness and that the former, together with TPS, might also be destructive and are intrinsically prone to divergent optimization. We propose, instead, to use an efficient and highly-expressive family of *diffeomorphisms* (*i.e.* differentiable invertible maps with a differentiable inverse) and demonstrate that this significantly improves regression and classification results on diverse tasks, while being more robust during training. The examples in Fig. 1 show that the affine model allows the network only to "zoom in" on the face, while the proposed models ("Diffeomorphic" and "Affine+Diffeomorphic") further stretch the face to become almost square. This intuitive "squarification" suffices to make standard CNNs match state-of-the-art performance.
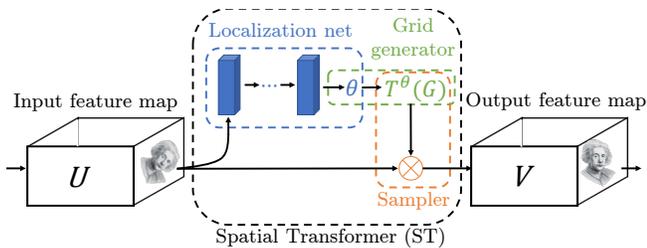
Figure 2: The spatial transformation layer [27].

This paper contributes the first diffeomorphic image transformations built into a deep neural network. This provides a simple layer that can be inserted into widely-used established architectures using only two lines of Tensor-Flow [1] code. Empirically, we find that this extra layer is sufficient to allow off-the-shelf neural networks to match state-of-the-art performance on several tasks. In *face verification* tasks, the interpretability of the diffeomorphic layer allows us to gain new insights into facial image analysis and see that a simple "squarification" transformation can significantly boost performance of a model. In the process of developing the diffeomorphic layer we gain further insights into traditional affine spatial transformers and show that their lack of invertibility causes divergent optimization.

2. Related work

Invariance is often key to success of computer-vision models. Traditionally, invariance to simple transformations (*translations*, *rotations*, *scalings*, *etc.*) was obtained by extracting invariant features; *e.g.*, *SIFT* features [39] are invariant to translations, rotations and scalings.

The reemergence of neural networks has shifted focus to *learned* features that are approximately invariant to key transformations. Convolutional layers learn filters that are applied in a translation-invariant fashion, but the filter response itself is not invariant. Max-pooling strategies alleviate this to some extent and provide invariance to small translations. In practice, however, pooling is done only over small regions (*e.g.*, 3×3), so each pooling layer provides an approximated spatial invariance of up to only a few pixels. Generalizing the convolution operator allows for further invariances; *e.g.*, Henriques *et al.* [23] show that invariance to two-parameter transformations is achievable at small computational cost. The *spatial transformer (ST)* layer [27] was introduced to allow for invariances to significantly-larger (parametrized) transformations. The ST-layer is the topic of the present paper and is further explored in § 3.1.

A complementary technique for achieving approximated invariance is *Data Augmentation*, where one artificially augments the training data with new samples created by transforming the original data via pre-specified transforma-

tions. This approach, however, requires knowing which transformations the model should be invariant to. This generally depends on the application domain [7, 28, 32, 37, 46]. The classic *Tangent Prop* [45] locally linearizes the known invariance, and forces the back-propagated gradient of a neural net to respect it. The linearization, however, implies that invariance is only infinitesimal. General linear invariances are also used for restricted Boltzmann machines [31, 49], but again the invariance is only infinitesimal. Note also that Hauberg *et al.* [22] argue that the specification of transformations in data augmentation should be viewed as a learning problem.

Pattern Theory [20] is *transformational* in the sense it focuses on transformations acting on objects rather than the objects themselves. Alternatively, all instantiations of a transformation type to which invariance is sought may be applied to each observation to produce orbits [19, 34], which can then be matched. While mathematically elegant, these approaches tend to be computationally expensive.

In computer vision, diffeomorphisms are used primarily in nonrigid registration and shape analysis (*e.g.* [5, 6, 8, 14, 21, 29, 38, 41, 56, 57, 62]). Since traditional approaches to diffeomorphisms require dire computational costs, several works tried to alleviate this, *e.g.* using control points [2, 15] or approximated/discretized diffeomorphisms [3, 58, 61]. Model complexity and computational concerns have, however, prevented the applicability of the ideas to large-scale image analysis. Due to these difficulties, it is unsurprising that diffeomorphisms were never explicitly incorporated within deep-learning architectures. This is unfortunate, especially as various authors have noted, in several beautiful theoretical papers, the potential benefits from linking diffeomorphisms to deep learning [4, 40, 48]. Note that while Yang *et al.* [60] use standard deep learning to *predict* diffeomorphisms (given diffeomorphism training-data), the latter are not part of the network itself. In contrast, our work is the first to explicitly incorporate diffeomorphisms (particularly, the *CPAB transformations* [17], which are both efficient and highly expressive) in deep-learning architectures.

3. Background

3.1. Spatial Transformer Layer

The Spatial Transformer (ST) [27] is a differentiable layer which applies a parametrized spatial transformation to a feature map during a single forward pass, where the the parameter, $\boldsymbol{\theta}$, depends on the feature map. From the user's standpoint, using an ST-layer is transparent as the latter works architecturally similarly to a conv-layer and can be inserted at any point of a CNN or a more complicated network (*e.g.*, recurrent neural networks [50]). The ST-layer consists of three parts, illustrated in Fig. 2 and listed below.

1. The localization network takes as input a feature

Optimizing non-invertible ST-layers is prone to instability
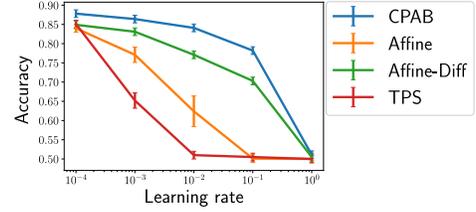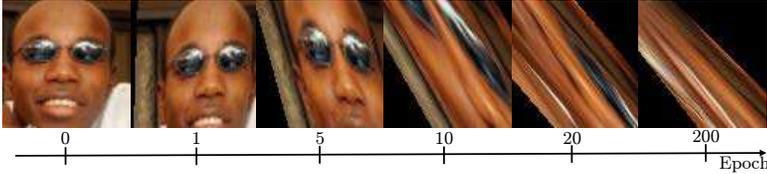
Figure 3: *Left:* Empirical inspection of the ST-layer, in the affine case, shows that transformations might become destructive (*i.e.*, singular or having a too-large condition number), so one cannot recover the untransformed image. The transformations we propose to use are invertible, so our approach does not have this drawback. *Right:* Additionally, the affine and TPS transformations seem to be more sensitive to the choice of learning rate in comparison with our proposed approach.

map U (*i.e.*, an input RGB image) to which it applies a regression network, f_{loc}, to produce a parameter, $\boldsymbol{\theta} = f_{\text{loc}}(\boldsymbol{U})$. This $\boldsymbol{\theta}$ corresponds to some $T^{\boldsymbol{\theta}} \in \mathcal{T}$ where $\mathcal{T}$ is a given parametrized family of transformations. Note that $d \triangleq \dim(\boldsymbol{\theta})$ depends on $\mathcal{T}$; *e.g.* for affine transformations, $\boldsymbol{\theta} \in \mathbb{R}^6$. f_{loc} may be of any regression network (*e.g.*, CNN) provided it ends with a fully-connected layer of d neurons.

2. The grid generator creates a grid $G \subset [-1, 1] \times [-1, 1]$ of evenly-spaced points of appropriate dimension. Each point in G is transformed by the transformation, $T^{\boldsymbol{\theta}}$.

3. The sampler computes the output feature map V by interpolating, using U, the values of V at $T^{\boldsymbol{\theta}}(G)$; *i.e.*, it performs image warping. The fact that it is preferable to formulate the latter via the inverse warp, $(T^{\boldsymbol{\theta}})^{-1}$ [53], is *one* reason why it is better to have an invertible $T^{\boldsymbol{\theta}}$ with an easily-computable inverse. In the context of ST-layers, the sampling kernel must be differentiable (as is the case, *e.g.*, for the bilinear kernel, which is the most popular choice).

In theory, the large expressiveness of the ST-layer stems from the fact that $\mathcal{T}$ can be chosen freely, with the constraint that the parametrization, $\boldsymbol{\theta} \mapsto T^{\boldsymbol{\theta}}$, must be differentiable w.r.t. $\boldsymbol{\theta}$, so the loss can back-propagate from the transformed sampled points, $\{T^{\boldsymbol{\theta}}(G_i)\}$, to the localization network f_{loc}. Jaderberg *et al.* [27] experiment with $\mathcal{T}$ being modeled as affine, plane projective, or a 16-point TPS, with the latter outperforming the others. While the affine transformations have gained most of the attention, recent STNs also consider homographies [35] and TPS maps [30, 35].

3.2. Spatial Transformations: Requirements

We note three requirements from the spatial transformations in order to train the localization network *robustly*:

1. Differentiability. As networks are trained by gradient-based methods, it is essential that the parametrized transformation, $T^{\boldsymbol{\theta}} : \mathbb{R}^2 \to \mathbb{R}^2$, $\boldsymbol{x} \mapsto T^{\boldsymbol{\theta}}(\boldsymbol{x})$, will be differentiable w.r.t. both $\boldsymbol{x}$ and $\boldsymbol{\theta}$; *e.g.*, this is the case for affine transformations, where $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 & \theta_5 & \theta_6 \end{bmatrix}^T \in \mathbb{R}^6$ and

$$T^{\boldsymbol{\theta}} : \mathbb{R}^2 \to \mathbb{R}^2, \quad T^{\boldsymbol{\theta}}(\boldsymbol{x}) \triangleq \begin{bmatrix} \theta_1 & \theta_2 \\ \theta_4 & \theta_5 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} \theta_3 \\ \theta_6 \end{bmatrix} . \quad (1)$$

Differentiability, w.r.t. $\boldsymbol{x}$, also implies that drastic *kinks* or *corners* in the transformed image are avoided.

2. Invertibility. During training, the stochastic gradient will, at times, point in directions that do *not* improve the performance. While this can help escaping local optima, it is often important that a subsequent step can revert back. If the ST-layer starts to predict nearly-singular maps, the back-propagated gradient will contain almost no information (see Fig. 3, left), making it hard to revert the optimization. Ensuring invertibility in the transformation avoids this failure mode. Both affine and TPS transformations may fail to be invertible. In the affine case, $f : \mathbb{R}^n \to \mathbb{R}^n$, $\boldsymbol{x} \mapsto A\boldsymbol{x} + \boldsymbol{b}$ (with: $A \in \mathbb{R}^{n \times n}$; $\boldsymbol{b} \in \mathbb{R}^n$) is invertible if and only if $\det A \neq 0$. Similarly, TPS is invertible if and only if the determinant of the TPS kernel is non-zero. Empirically, we observe (Fig. 3, right) that invertible transformation families are less sensitive to the choice of learning rate. Particularly, invertible families (*blue*; *green*) are significantly more robust than those that do not ensure invertibility (*yellow*; *red*). Moreover, the invertibility avoids destructive *folds* in the transformed images.

3. Having a differentiable inverse. In order to take a gradient step that reverts an unfortunate previous step, the derivative of the inverse transformation should also exist.

Together, these three requirements coincide with the definition of a *diffeomorphism*: a (C^1) *diffeomorphism* is a differentiable invertible map with a differentiable inverse. Transferring these observations into a practical implementation, however, presents algorithmic and mathematical challenges as evaluations of general diffeomorphisms, let alone their derivatives, tend to be computationally demanding. Note that due to the possible lack of invertibility, *neither affine nor TPS maps are guaranteed to be diffeomorphisms*.

3.3. Diffeomorphisms

We seek a family of diffeomorphisms, parameterized by $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 & ... & \theta_d \end{bmatrix}^T \in \mathbb{R}^d$, that is highly expressive, easy to implement, and computationally efficient. We first discuss a family that meets only the last two requirements.

Affine diffeomorphisms. Let $\widetilde{\boldsymbol{x}} \triangleq \begin{bmatrix} \boldsymbol{x}^T & 1 \end{bmatrix}^T \in \mathbb{R}^3$ de-
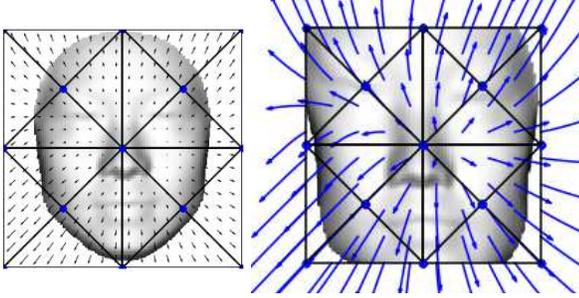
Figure 4: An example of a CPAB transformation. *Left:* A [2,2]-tessellation of the domain is overlaid on an image of face [10]. The black arrows defines a velocity field that is continuous piecewise-affine w.r.t. the tessellation. *Right:* Integrating the velocity field generates paths (blue) that define a diffeomorphic image deformation.

note the homogeneous-coordinate representation of $\boldsymbol{x} \in \mathbb{R}^2$. An affine map $\boldsymbol{x} \mapsto T^{\boldsymbol{\theta}}(\boldsymbol{x})$ may be written as

$$\mathbb{R}^3 \ni \begin{bmatrix} T^{\boldsymbol{\theta}}(\boldsymbol{x}) \\ 1 \end{bmatrix} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \\ 0 & 0 & 1 \end{bmatrix} \widetilde{\boldsymbol{x}}. \quad (2)$$

This map is invertible if and only if the matrix above is invertible. In which case, its inverse map is also affine, hence differentiable. Thus, *invertible* affine maps are diffeomorphisms. A way to get invertibility is as follows. Let $\boldsymbol{v}^{\boldsymbol{\theta}} : \mathbb{R}^2 \to \mathbb{R}^2$ be an affine *velocity field* given by

$$\boldsymbol{v}^{\boldsymbol{\theta}} : \boldsymbol{x} \mapsto A\widetilde{\boldsymbol{x}} \quad \text{where} \quad A \triangleq \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \end{bmatrix} \in \mathbb{R}^{2\times3}. \quad (3)$$

Let $\mathbf{expm} : \mathbb{R}^{3\times3} \to \mathbb{R}^{3\times3}$ denote the matrix exponential. If we redefine $T^{\boldsymbol{\theta}} : \mathbb{R}^2 \to \mathbb{R}^2$ via

$$\begin{bmatrix} T^{\boldsymbol{\theta}}(\boldsymbol{x}) \\ 1 \end{bmatrix} = \mathbf{expm}(\tilde{A})\widetilde{\boldsymbol{x}}, \quad \tilde{A} \triangleq \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \\ 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{3\times3}, \quad (4)$$

then (since the last row of $\mathbf{expm}(\tilde{A})$ is $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$ and $\det \mathbf{expm}(\tilde{A}) > 0$) $T^{\boldsymbol{\theta}}$ is an affine diffeomorphism. Moreover, it can be shown that this transformation is the solution, $\phi : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$, to the *integral equation*

$$\phi^{\boldsymbol{\theta}}(\boldsymbol{x}; 1) = \boldsymbol{x} + \int_0^1 \boldsymbol{v}^{\boldsymbol{\theta}}(\phi^{\boldsymbol{\theta}}(\boldsymbol{x}; \tau)) \, \mathrm{d}\tau. \quad (5)$$

Thus, affine velocity fields yield affine diffeomorphisms.

More flexible diffeomorphisms. Equation 5 gives the interpretation that the velocity field is an infinitesimal transformation, applied repeatedly through integration. This suggests that flexible diffeomorphisms can be designed by considering richer velocity fields. A key advantage to defining diffeomorphisms via velocity fields is that while a diffeomorphism family is always a nonlinear space, its corresponding velocity-field family is usually a linear space [57]. Thus, to have a highly-expressive diffeomorphism family, we seek a highly-expressive linear space of velocity fields

$\mathcal{V}$, which can then be integrated according to Eq. 5. Once we move beyond the affine case, integral equations usually lack analytic solutions, so one often resorts to numerical methods with a trade-off between computation time and accuracy. A natural question is how to pick $\mathcal{V}$. In general, there is a also trade-off between the expressiveness of $\mathcal{V}$ on the one hand, and keeping both d and the computational cost (*e.g.*, of solving Eq. 5) low on the other hand.

4. Diffeomorphic Transformer Layers

4.1. CPA-Based Transformations

In our context, as we propose incorporating diffeomorphisms in a deep-learning architecture (via an ST-layer), the aforementioned computation time is even more important than in traditional computer-vision applications of diffeomorphisms. This is because during training, evaluations of $T^{\boldsymbol{\theta}}(\boldsymbol{x})$, as well as the gradient $\nabla_{\boldsymbol{\theta}} T^{\boldsymbol{\theta}}(\boldsymbol{x})$, are computed at multiple pixel locations $\boldsymbol{x}$ for multiple $\boldsymbol{\theta}$'s. Thus, explicit incorporation of highly-expressive diffeomorphism families into deep-learning architectures used to be infeasible.

Recently, however, Freifeld *et al*. proposed the CPAB transformations [17, 18], which offer a happy medium between expressiveness and efficiency. This makes CPAB transformations a natural choice in a deep-learning context.

The name of the CPAB (CPA-Based) transformations stems from the fact that they are based on the integration of Continuous Piecewise-Affine (CPA) velocity fields. The term "piecewise" is w.r.t. some tessellation of the image domain into cells. For a nominal tessellation, the corresponding space of CPAB transformation is given by

$$\mathcal{T} \triangleq \{T^{\boldsymbol{\theta}} : \boldsymbol{x} \mapsto \boldsymbol{\phi}^{\boldsymbol{\theta}}(\boldsymbol{x}, 1) \text{ s.t. } \boldsymbol{\phi}^{\boldsymbol{\theta}}(\boldsymbol{x}, t) \text{ solves Eq. } 5\} \quad (6)$$

where $\boldsymbol{v}^{\boldsymbol{\theta}}$ is CPA w.r.t. the tessellation; see Fig. 4. CPA velocity fields support an integration method that is both faster and more accurate than typical integration methods [17].

The fineness of the tessellation controls the trade-off between expressiveness on the one hand and computational complexity and dimensionality on the other hand. In our context, the following is important to note.

Flexibility in expressiveness. If more expressive transformations are needed, a finer tessellation can be chosen at the cost of computation speed and a higher-dimensional d.

Low-dimensional representation. Finer tessellations imply a higher $d = \dim(\boldsymbol{\theta})$. However, one can get fairly-expressive transformations even with a relatively-low d; *e.g.*, in our experiments $d = 58$. In [12] we investigate the choice of the tessellation size, but keep it fixed in the remainder of the paper.

Initialization. Since $\boldsymbol{\theta} = \boldsymbol{0}$ gives the identity map we initialize the CPAB layer by setting all the weights in the final layer of the localization net to zero.

Dataset	Img. dim	# of classes	Training samples	Test samples	Base architecture	# of network parameters
Distorted MNIST	(42,42,1)	10	50000	10000	[27]	400K
Fashion MNIST	(28,28,1)	10	50000	10000	[27]	400K
CIFAR10	(32,32,3)	10	50000	10000	[54]	1M
CIFAR100	(32,32,3)	100	50000	10000	[54]	1M
LFW (restricted)	(250,250,3)	2	5600	400	[51]	200K
CelebA	(218,178,3)	40	192469	10130	[63]	138M

Table 1: The different datasets used for comparing the models.

4.2. The Proposed CPAB and Affine+CPAB Layers

In this paper we propose two novel network layers:

CPAB layer: Our main proposal is replacing the affine transformations with the diffeomorphic CPAB transformations. This raises some technical challenges in implementing CPAB transformations effectively in a deep-learning framework; see § 4.2.1 for more details.

Affine+CPAB layer: An oft-used modeling paradigm is to first use a simple model (*e.g.*, affine) to do a crude estimation and then use a more complicated model for the fine estimation. We propose a similar approach, with the ST-Affine layer being the simpler model and our proposed CPAB layer being the more complicated one. Thus we propose an ST-Affine+CPAB layer with two localization nets (one for each transformation) in serial: the first uses affine transformations to do a rough alignment and then the second uses CPAB transformations for refinement.

4.2.1   Implementation

We implemented the CPAB transformations and gradient in CUDA within TensorFlow's C++ API, and the rest of the code within TensorFlow's python API. Using our code[1] requires only two lines in TensorFlow: one for setting the tessellation and one for incorporating the transformations into TensorFlow's network graph.

CPAB transformation evaluations (*i.e.*, evaluating $\boldsymbol{x} \mapsto T^{\boldsymbol{\theta}}(\boldsymbol{x})$) do not lend themselves to efficient pure-TensorFlow implementation; this is due to two main reasons (mentioned briefly here but see [12] for more details). The first is that CPAB evaluations require repeated cell indexing. In the computational-graph paradigm, employed by TensorFlow, this is translated to numerous redundant evaluations. The second is graph construction. The evaluations also require iterative application of the transformation. The resulting computational graph turns out to be inefficient, possibly since currently TensorFlow is not optimized for such cases.

Having implemented CPAB evaluations in both CUDA and pure TensorFlow, we empirically found that the latter is 11 times slower than the former for forward passes and 5 times slower for backward passes.

The CPAB gradient, $\partial T^{\boldsymbol{\theta}}(\boldsymbol{x})/\partial\boldsymbol{\theta}$, whose mathematical details can be found in [16], does not have a closed-form expression. Rather, it is given only via the solution of a system of coupled integral equations [18]. As such, TensorFlow's auto-differentiation is inapplicable for computing it.

Finally, the CUDA implementation from [17, 18] lacked TensorFlow interface (as it was not designed for DL architectures) and was also slower than our new implementation. First, here we derived closed-form expressions for the associated matrix exponentials. Second, we simplified the CPA integration algorithm. Third, we added two additional parallelism levels; while in [17, 18] parallelization was done only over different pixels, here we also parallelize over different images as well as the d components of the CPAB gradient. Without these speedups – explained in more detail in [12] – CPAB layers would have been impractical.

5. Experimental Results and Discussion

We evaluated the accuracy of the proposed transformer models and compared performance with both other transformer models and a standard CNN. The evaluation was done on several datasets, in both small and large scale.

Datasets: We used 6 different datasets, listed in Table 1. On all 6 datasets, we trained 7 CNNs: 1) without an ST-layer; 2) with an ST-Affine layer; 3) with an ST-AffineDiff layer; 4) with an ST-Homography layer; 5) with an ST-TPS (16 point) layer; 6) with an ST-CPAB layer; 7) with an ST-Affine+CPAB layer. The base architectures, chosen based on recent work on the datasets [27, 51, 54, 63], represent common and widely-used archetypal architectures, rather than highly-customized state-of-the art work for specific tasks. For all transformer models, we put the ST-layer right after the input layer, just before the feature extraction layers. For a fair and consistent comparison, the number of parameters is kept the same for all networks operating on a given dataset. Thus networks with ST-layers have fewer parameters in their feature extraction part because some parameters are used in the localization network.

5.1. Distorted MNIST

We started with a control experiment to verify and understand the proposed layers. We trained the 7 networks to classify MNIST digits that had been distorted in two

---

[1]Our code is available at: github.com/SkafteNicki/ddtn

Model	MNIST Distortion		Training
	RTS	CPAB	time [s]
FCN	0.945	0.918	280
CNN	0.974	0.952	360
ST-Affine FCN	0.986	0.964	400
ST-Affine CNN	0.992	0.974	420
ST-CPAB FCN	0.980	0.980	11587
ST-CPAB CNN	0.982	0.989	12067
ST-Affine+CPAB CNN	**0.996**	**0.993**	13058

Table 2: Classification accuracy on the two distorted MNIST datasets. All models have the same training settings: batch size 100; learning rate 0.0001; Adam optimizer; no weight decay; no dropout; 100 epochs. Results are averages of 5 runs with random initializations.

different ways: random rotation+scale+translation (RTS); random CPAB transformation+translation (CPAB). This is akin to an experiment from [27], except here we also consider CPAB transformations. Thus, we have designed the RTS dataset to match the ST-Affine layer and the CPAB dataset to match the ST-CPAB layer. Additionally, we also tested the effect of modeling the localization network as either a fully-connected network (FCN) or a CNN. Figure 5 shows 3 samples from each of the distorted datasets, after being transformed by the networks. Table 2 shows classification accuracy attained by the models. The best-performing model for both datasets is the CNN that has both an affine and a CPAB layer. This is supported by the figure, which shows that samples for this model are fully centered and zoomed in, making them easier to classify. The second-best model in the RTS case is the ST-Affine CNN and for the CPAB dataset it is the ST-CPAB CNN; this is unsurprising as we have tailored the datasets to fit the different transformations. This result also suggests there is no single transformer model that will be best for all datasets. The transformations learned by the ST-Affine CNN and ST-CPAB CNN have a similar effect on the images, since they either zoom in on the digits or expand them. The fact that the transformer models outperformed the no-transformer suggests it is usually beneficial to allocate some parameters to the localization network, thus getting "optimized" samples before the classification network, as opposed to using a slightly larger network with no ST layers.

We additionally find that networks whose localization net uses conv-layers (as opposed to FC-layers) perform consistently better, in agreement with similar findings in [27]. In other words, the localization nets utilize spatial information in the images to predict better transformations.

A current disadvantage of the proposed layers is the longer computation time during training. Although we achieve low computation time per transformation in comparison with other (unrelated-to-deep-learning) implemen-

Figure 5: Examples of samples from the generated distorted MNIST datasets, using the different models, right before they are fed into the feature-extraction layer.

tations of diffeomorphisms (including [17, 18]), it is still higher than for an affine spatial transformer. This is due to the added complexity of the transformation. That said, once the training stage is over, the prediction of new samples is only 5% slower than competing models.

5.2. MNIST, Fashion MNIST and CIFAR

The second experiment set was performed on the original MNIST dataset, the Fashion MNIST [59] dataset, and the CIFAR10 and CIFAR100 datasets. Table 3 summarizes the results, and a deformed sample from each dataset is shown in Fig. 7. For the MNIST dataset we observe an improvement using transformer layers, which can be explained by the zooming/expanding effect we observed earlier.

For the Fashion MNIST dataset, however, we see a small drop in accuracy. By visual inspection, the objects are already in focus, and it is therefore unnecessary to transform the samples; *i.e.*, this is not the right choice of model for this dataset, since we are "wasting" parameters in the localization network on predicting the identity transformation. We experimented with increasing the number of parameters in the classification network of the ST-CPAB model to the same number as the no-transformer model, and we find that the performance is the same as that of the no-transformer network. In other words, the ST-layers did not hurt the performance. Figure 7 also shows that the transformation models can introduce interpolation artifacts. This effect, which might remove key features (*e.g.*, see the Nike logo), is particularly present in the ST-Affine+CPAB model, as the image was interpolated twice. This suggests that images must have a certain minimal resolution when using ST-layers.

In both the CIFAR experiments, we observe a small performance gain due to the ST-layers. In these datasets, some objects are in focus while some are not. The ST-layers apply transformations that zoom in on the objects not in focus.

Model	Dataset	
	MNIST	Fashion MNIST
No transformer	0.991	**0.922**
ST-Affine	0.993	0.919
ST-AffineDiff	0.994	0.920
ST-Homography	0.993	0.919
ST-TPS	0.996	0.918
ST-CPAB	0.994	0.917
ST-Affine+CPAB	**0.996**	0.913
	CIFAR10	CIFAR100
No transformer	0.870	0.642
ST-Affine	0.891	0.653
ST-AffineDiff	0.892	0.654
ST-Homography	0.891	0.653
ST-TPS	0.893	0.656
ST-CPAB	**0.895**	**0.659**
ST-Affine+CPAB	0.889	0.652
	LFW	CelebA
No transformer	0.788	0.712
ST-Affine	0.840	0.734
ST-AffineDiff	0.842	0.740
ST-Homography	0.843	0.742
ST-TPS	0.851	0.751
ST-CPAB	0.878	0.756
ST-Affine+CPAB	**0.893**	**0.772**

Table 3: Classification performance of the CNN models trained, with or without transformations layers, on the datasets from Table 1. All models were trained using the same settings: (batch size 100; learning rate 0.0001; Adam optimizer; no weight decay; no dropout). For different datasets we used different numbers of epochs, but on each dataset, the number of epochs was the same for all models.

Both the CPAB and the affine ST-layers have learned similar transformations. Again we observe the interpolation issue: it is not beneficial to use the Affine+CPAB model, since the image gets over-smoothed.

5.3. Restricted LFW and CelebA

The results of the previous experiments suggest that in a more challenging dataset, which also has higher resolution, we are likely to see a more substantial gain from using advanced transformations. The next experiment set was therefore performed on two facial datasets, each with different tasks. For the relatively-small Labeled-Face-in-the-Wild (LFW) dataset [24], we trained a Siamese network [9] for face verification (binary classification task). We here worked in the "restricted" setting of the LFW dataset, where our findings were based on the mean accuracy in the "View

2" ten splits (cf. [24] for details). Next, we worked with the large CelebA dataset where we predicted 40 binary attributes (big nose, male, smiling, wearing hat, etc.) based on facial images. Table 3 shows the results. We observe a clear difference between the different models. The results can be explained by looking at some deformed examples, see Figs. 1 and 7: the ST-layers zoom in on the important part of the face. The CPAB model does an additional "squarification" of the face removing unimportant information in the corners of the image. The combined Affine+CPAB model inherits the effect of both layers, first an initial zooming of the affine layer and then a squarification of the face, such that only important facial features are preserved. The high resolution of the images prevents over-smoothing.

Figures 6a and 6b show training and test accuracy for the LFW experiments. Inspecting the curves, the model with the highest training accuracy achieves the lowest test accuracy and vice versa. The transformer models take more epochs to train, but eventually outperform the others.

By inspection of the deformed images, the learned transformations are similar for all input samples. This can be explained by the limited size of the localization network (2 conv-layers followed by 1 FC-layer), which restricts the diversity of the transformations that the localization network can predict. We have investigated this behaviour for the LFW dataset, using Principal Component Analysis of the predicted θ values on the test set. In Fig. 6c we have plotted variations of the two leading principal components. We see that these mainly contribute a vertical and a horizontal translation of the faces. Thus, the features that the localization network is extracting from the images are used to determine the center of the face in the image, which the transformation then zooms in on.

5.4. Unrestricted LFW

In all the experiments until now, in order to get a clear picture of the different performances obtained by the different models, we have intentionally avoided using commonly-used deep-learning tricks (e.g.: data augmentation; dropout). In our last experiment, however, we took advantage of such tricks to train a deeper network on the LFW dataset in the unrestricted setting. In this setting we know the identity of each image, thus we could form new pairs. We formed 50K positive pairs and 50K negative pairs, and by using data augmentation of each sample (random rotate, translate, flip left-right) we generated a total of 400K training samples. We compared our results to the state-of-the-art in the "unrestricted-label free outside data" category, which is the closest to our setting. We match state-of-the-art results from Ding et al. [13], who use a manually-designed facial image descriptor, on which they train two different classifiers, combined with a linear SVM. In other words, our proposed end-to-end learning model obtains similar perfor-
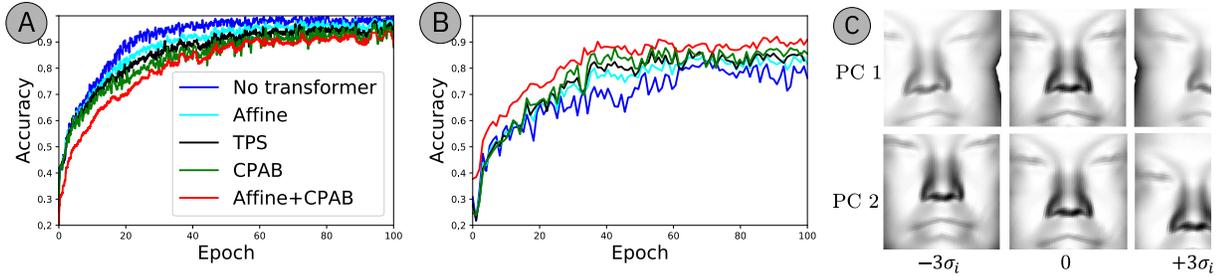
Figure 6: Training (A) and test (B) accuracy for the LFW experiments. While the transformer nets are slower to converge than the no-transformer net, they eventually reach better performance. (C) PCA of predicted $\boldsymbol{\theta}$ values from the localization network. By varying PC 1 and 2, we observe that these capture mainly horizontal and vertical translation of the transformations. This corresponds to the small variations in the facial center of the people in the LFW dataset.

	Test acc. $\hat{\mu} \pm S_E$
CNN (no transformer)	0.8930 ± 0.0028
ST-Affine	0.9129 ± 0.0032
ST-AffineDiff	0.9145 ± 0.0029
ST-Homography	0.9139 ± 0.0031
ST-TPS	0.9218 ± 0.0032
ST-CPAB	0.9368 ± 0.0035
ST-Affine+CPAB	0.9543 ± 0.0046
Ding *et al.* [13]	$\mathbf{0.9558 \pm 0.0034}$
Zhu *et al.* [65]	0.9525 ± 0.0036

Table 4: Results in the unrestricted LFW setting.

mance to their customized feature-engineered model. We match another customized method, from Zhu *et al.* [64], that uses a 3D pose and expression model trained on outside data to transform the input images. We achieve slightly better performance using a simple learned 2D transformation as part of a simple deep-learning pipeline.

6. Conclusion

We have shown that highly-expressive diffeomorphisms are both useful and practical in deep-learning pipelines; particularly, we have shown that extending the traditional ST-layers [27] to support diffeomorphic CPAB transformations [17, 18] leads to performance gains on established benchmarks and matches state-of-the-art on the (unrestricted) LFW dataset. Notably, our generic 2D transformations outperform transformations driven by nontrivial 3D face models on the LFW dataset. The learned transformations are interpretable and suggest that in facial image analysis, a simple image "squarification" can improve performance. We also find that diffeomorphism have good optimization properties, *e.g. diffeomomorphic* affine transformations lead to more robust optimization and better empirical performance than more general affine transformations. As using diffeomorphic affine transformations is easy, there is little reason to consider non-diffeomorphic ones. Our
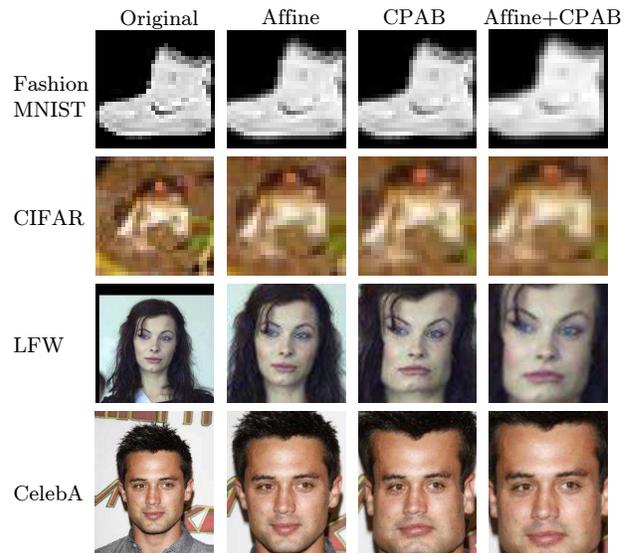
Figure 7: Examples of learned transformations for the different models on the different datasets. For more transformed samples, see [12].

public code[1] is easy to use within standard deep-learning software: only two lines of additional code are needed to extend an existing model. The proposed models can be extended in several way. First, while we have focused on 2D, CPAB transformations are also applicable in, *e.g.*, 1D and 3D. Second, the ST-layers may be inserted in different locations in the network, not just after the input. Finally, it may be fruitful to consider multiple ST-layers acting in parallel to allow for multiple prototype transformations.

References

[1] M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from http://tensorflow.org. 2

[2] S. Allassonniere, S. Durrleman, and E. Kuhn. Bayesian mixed effect atlas estimation with a diffeomorphic deformation model. *SIAM Journal on Imaging Sciences*, 2015. 2

[3] S. Allassonnière, A. Trouvé, and L. Younes. Geodesic shooting and diffeomorphic matching via textured meshes. In *EMMCVPR*. Springer, 2005. 2

[4] F. Anselmi, L. Rosasco, and T. Poggio. On invariance and selectivity in representation learning. *Information and Inference: A Journal of the IMA*, 2016. 2

[5] V. Arsigny, O. Commowick, X. Pennec, and N. Ayache. A log-euclidean framework for statistics on diffeomorphisms. In *MICCAI*. 2006. 2

[6] V. Arsigny, O. Commowick, X. Pennec, and N. Ayache. A log-euclidean polyaffine framework for locally rigid or affine registration. In *BIR*. Springer, 2006. 2

[7] H. S. Baird. Document image defect models. In *SDIA*. Springer, 1992. 2

[8] M. F. Beg, M. I. Miller, A. Trouvé, and L. Younes. Computing large deformation metric mappings via geodesic flows of diffeomorphisms. *IJCV*, 2005. 2

[9] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005. 7

[10] C. Creusot. Automatic 3d face landmarking. Software available from http://www.clementcreusot.com/phd/. 4

[11] A. Desmaison. The power of spatial transformer networks, september 2015. http://torch.ch/blog/2015/09/07/spatial_transformers.html. 1

[12] N. S. Detlefsen, O. Freifeld, and S. Hauberg. Deep diffeomorphic transformer networks – supplemental material. In *CVPR*, 2018. 4, 5, 8

[13] C. Ding, J. Choi, D. Tao, and L. S. Davis. Multi-directional multi-level dual-cross patterns for robust face recognition. *IEEE transactions on PAMI*, 2016. 7, 8

[14] P. Dupuis, U. Grenander, and M. I. Miller. Variational problems on flows of diffeomorphisms for image matching. *QAM*, 1998. 2

[15] S. Durrleman, S. Allassonnière, and S. Joshi. Sparse adaptive parameterization of variability in image ensembles. *IJCV*, 2013. 2

[16] O. Freifeld. Deriving the CPAB derivative. Technical report, The Department of Computer Science, Ben-Gurion University, 2018. 5

[17] O. Freifeld, S. Hauberg, K. Batmanghelich, and J. W. Fisher. Highly-expressive spaces of well-behaved transformations: Keeping it simple. In *ICCV*, 2015. 2, 4, 5, 6, 8

[18] O. Freifeld, S. Hauberg, K. Batmanghelich, and J. W. Fisher. Transformations based on continuous piecewise-affine velocity fields. *IEEE TPAMI*, 2017. 4, 5, 6, 8

[19] T. Graepel and R. Herbrich. Invariant pattern recognition by semi-definite programming machines. In S. Thrun, L. Saul, and B. Schölkopf, editors, *NIPS*. MIT Press, 2004. 2

[20] U. Grenander. *General pattern theory: A mathematical study of regular structures*. Clarendon Press, 1993. 2

[21] H. Guo, A. Rangarajan, and S. Joshi. Diffeomorphic point matching. In *Handbook of Mathematical Models in Computer Vision*. Springer, 2006. 2

[22] S. Hauberg, O. Freifeld, A. B. L. Larsen, J. W. F. III, and L. K. Hansen. Dreaming more data: Class-dependent distributions over diffeomorphisms for learned data augmentation. In *Proceedings of the 19th international Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 51, pages 342–350, 2016. 2

[23] J. F. Henriques and A. Vedaldi. Warped convolutions: Efficient invariance to spatial transformations. In *ICML*, 2017. 2

[24] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. 7

[25] J. Huang and K. Murphy. Efficient inference in occlusion-aware generative models of images. *arXiv preprint arXiv:1511.06362*, 2015. 1

[26] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. *arXiv preprint arXiv:1406.2227*, 2014. 1

[27] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *NIPS*, 2015. 1, 2, 3, 5, 6, 8

[28] N. Jaitly and G. E. Hinton. Vocal tract length perturbation (VTLP) improves speech recognition. In *ICML*, 2013. 2

[29] S. C. Joshi and M. I. Miller. Landmark matching via large deformation diffeomorphisms. *IEEE TIP*, 2000. 2

[30] A. Kanazawa, D. W. Jacobs, and M. Chandraker. Warpnet: Weakly supervised matching for single-view reconstruction. In *CVPR*, 2016. 3

[31] J. J. Kivinen and C. K. Williams. Transformation equivariant boltzmann machines. In *ICANN*. Springer, 2011. 2

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 2

[33] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998. 1

[34] Q. Liao, J. Z. Leibo, and T. Poggio. Learning invariant representations and applications to face verification. In *NIPS*, 2013. 2

[35] C. Lin and S. Lucey. Inverse compositional spatial transformer networks. *CoRR*, abs/1612.03897, 2016. 1, 3

[36] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1

[37] G. Loosli, S. Canu, and L. Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, 2007. 2

[38] M. Lorenzi and X. Pennec. Geodesics, parallel transport & one-parameter subgroups for diffeomorphic image registration. *IJCV*, 2012. 2

[39] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 2004. 2

[40] S. Mallat. Understanding deep convolutional networks. *Phil. Trans. R. Soc. A*, 2016. 2

[41] M. Nielsen, P. Johansen, A. Jackson, B. Lautrup, and S. Hauberg. Brownian warps for non-rigid registration. *Journal of Mathematical Imaging and Vision*, 31:221–231, 2008. 2

[42] D. Rezende, I. Danihelka, K. Gregor, D. Wierstra, et al. One-shot generalization in deep generative models. In *ICML*, 2016. 1

[43] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*. Springer, 2015. 1

[44] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *CVPR*, 2015. 1

[45] P. Simard, B. Victorri, Y. LeCun, and J. S. Denker. Tangent prop – a formalism for specifying selected invariances in an adaptive network. In *NIPS*, 1992. 2

[46] P. Y. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*. IEEE Computer Society, 2003. 2

[47] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[48] S. Soatto and A. Chiuso. Visual representations: Defining properties and deep approximations. *arXiv preprint arXiv:1411.7676*, 2014. 2

[49] K. Sohn and H. Lee. Learning invariant representations with local transformations. In *ICML*, 2012. 2

[50] S. K. Sønderby, C. K. Sønderby, L. Maaløe, and O. Winther. Recurrent spatial transformer networks. *arXiv preprint arXiv:1509.05329*, 2015. 2

[51] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. In *NIPS*, 2014. 5

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 1

[53] R. Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010. 3

[54] Tensorflow. Convolutional neural networks. From https://www.tensorflow.org/tutorials/deep_cnncifar-10_model. 5

[55] J. J. Tompson, A. Jain, Y. LeCun, and C. Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *NIPS*, 2014. 1

[56] A. Trouvé. Diffeomorphisms groups and pattern matching in image analysis. *IJCV*, 1998. 2

[57] M. Vaillant, M. I. Miller, L. Younes, and A. Trouvé. Statistics on diffeomorphisms via tangent space representations. *NeuroImage*, 2004. 2, 4

[58] T. Vercauteren, X. Pennec, A. Perchant, and N. Ayache. Diffeomorphic demons: Efficient non-parametric image registration. *NeuroImage*, 2009. 2

[59] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017. 6

[60] X. Yang, R. Kwitt, M. Styner, and M. Niethammer. Quicksilver: Fast predictive image registration–a deep learning approach. *NeuroImage*. 2

[61] M. Zhang and P. T. Fletcher. Finite-dimensional Lie algebras for fast diffeomorphic image registration. In *IPMI*, 2015. 2

[62] M. Zhang and P. T. Fletcher. Bayesian statistical shape analysis on the manifold of diffeomorphisms. In *Algorithmic Advances in Riemannian Geometry and Applications*. 2016. 2

[63] N. Zhang, M. Paluri, M. Ranzato, T. Darrell, and L. Bourdev. Panda: Pose aligned networks for deep attribute modeling. In *ICCV*, 2014. 5

[64] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros. View synthesis by appearance flow. *CoRR*, abs/1605.03557, 2016. 8

[65] X. Zhu, Z. Lei, J. Yan, D. Yi, and S. Z. Li. High-fidelity pose and expression normalization for face recognition in the wild. In *CVPR*, 2015. 8