# 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

*This document contains supplementary material for the CVPR 2018 paper "Deep Diffeomophic Transformer Networks". We use the same notation as the in the paper [2]. The current documents contains the following sections: 1. Implementation details; 2. Tessellation size; 3. Analytic matrix exponential for 3x3 matrices; 4. Transformed samples.*

## 1. Implementation details

In the following section we present pseudocode for computing and implementing the proposed ST-CPAB layer, together with the computational issues associated with implementing it in the TensorFlow framework.

As stated in the paper, incorporating the proposed layer into existing network architectures is simple, and requires merely two lines of code. The first line sets up $\boldsymbol{B}$ (the basis for the space of the CPA velocity fields); this is done using Algorithm 1 below [2]. This computation is done only once.

**Algorithm 1** Constructing the basis $\boldsymbol{B}$. The code involves finding all shared vertices $V$ between cells in the tessellation, constructing the constrain matrix $\mathbf{L}$ that encodes continuity constrains and then finally finding the basis $\boldsymbol{B}$ as the null space of $\mathbf{L}$; see [2] for more details.

**Require:** Number of the square tessellation cells in the x,y directions: $ncx, ncy$ (the final number of squares will be $4 \times ncx \times ncy$ as each square is divided into 4 triangles).

1: $V$ = find_tessellation_vertices($ncx$, $ncy$)
2: $\mathbf{L}$ = create_continuity_constraints($V$) // encodes the continuity constraints, which are linear (since the tessellation's cells are convex polytopes [3]) as a matrix
3: $\boldsymbol{B}$ = null($\mathbf{L}$) // Finds the null space of the matrix $\mathbf{L}$
4: **return** $\boldsymbol{B}$

Most of the computation time of the ST-CPAB layer is spent on evaluating the CPAB transformations; *i.e.*, transforming the grid points $G$ (see the discussion on the ST-layer in the paper). Pseudocode for this is given in Algorithm 2. Note Algorithm 2 is a simplification of the corresponding algorithm from Freifeld *et al*. [2]. In the original implementation there is an extra line after line 9, that checks whether $idx_t$ equals to $idx_{t-1}$; *i.e.*, have we moved from one cell to another. If so, Freifeld *et al*. called a generic ODE solver to get a better estimate of the cell boundary. By empirical testing, however, we found that the numerical error introduced by removing this extra check is negligible, and in the end speeds up the computation since we avoid thread divergence.

**Algorithm 2** CPAB transformer. The code integrates the velocity field $\boldsymbol{v}^{\boldsymbol{\theta}}$, by iteratively applying affine cell-dependent transformations $T_i$ that are computed from the fixed basis $\boldsymbol{B}$ and the parameter $\boldsymbol{\theta}$. See [2, 3] for more details.

**Require:** Number of tessellation cells in the x,y directions: $ncx, ncy$, Basis $\boldsymbol{B}$, Grid $G = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$, Parametrization $\boldsymbol{\theta}$.

$\tilde{A} = \boldsymbol{B}\boldsymbol{\theta}$
$A = vec^{-1}(\tilde{A})$
**for** i = 1:(4*ncx*ncy) **do in parallel**
    $T_i = \textbf{expm}(A_i)$
**for** $\boldsymbol{x}_i \in G$ **do in parallel**
    $\boldsymbol{x}_{0,i} = \boldsymbol{x}_i$
    **for** t = 1:50 **do**
        $idx_{t-1}$ = findcellidx($\boldsymbol{x}_{t-1,i}$ , ncx, ncy)
        $\boldsymbol{x}_{t,i} = T_{idx_{t-1}} \boldsymbol{x}_{t-1,i}$
    $G_{trans} = \{\boldsymbol{x}_{50,1}, \boldsymbol{x}_{50,2}, ..., \boldsymbol{x}_{50,n}\}$
**return** $G_{trans}$

Algorithm 2 is in principle simple to compute: just iteratively find the cell index of the points and apply the matrix belonging to that cell. However, it is worth mentioning that this simple code does not suit the computational paradigm by TensorFlow at the moment, due to two reasons:

1. **Cell indexing**: The findcellidx-function mostly consists of simple modulus and division operations. However, because the transformation needs to extend beyond the image domain, we need to check 5 different cases (point to the right, left, above, below our domain or inside domain). Even though each case consists of only a few simple operations, in a computational-graph paradigm we need to evaluate all conditions for all points because we cannot know, a-priori, which cell we are in. Thus, a naive TensorFlow implementation ends up performing a number of operations which is approximately 5x the number that is needed.

2. **Graph construction**: To evaluate the transformation, we iteratively apply the set of transformations $T_i$ 50 times in a static way to all points. Thus, in pure TensorFlow, a single for-loop creates 50 nodes in the computational graph, which heavily decreases performance. Although TensorFlow's API supports looped operations (tf.while_loop(...)) it does not appear to be optimized for such use cases at the current moment.

In general we found that a pure TensorFlow implementation of Algorithm 2 is 11 times slower than our CUDA implementation for forward passes and 5 times slower for backward passes.

With the implementation of the CPAB transformer, it is easy to define the CPAB-layer, see Algorithm 3. As in any other ST-layer, this step is associated with a grid generator and an interpolation method (bilinear in most cases).

**Algorithm 3** CPAB layer. The final CPAB layer is a concatenation of the CPAB transformer in Algorithm 2 with a grid generator (in our case a simple meshgrid of points) and a bilinear interpolation kernel.

**Require:** Batch of images $im$, Batch of parameters $\boldsymbol{\theta}$, Output size of each interpolated image $(out_w, out_h)$, Basis $\boldsymbol{B}$.

$G$ = meshgrid($out_w$, $out_h$)
$G_{trans}$ = CPAB_transformer($G$, $\boldsymbol{\theta}$, $\boldsymbol{B}$)
$im_{out}$ = interpolate($im$, $G_{trans}$, $(out_w, out_h)$)
**return** $im_{out}$

Algorithm 3 describes the forward pass of the operations. For the backwards pass, we compute the gradient of the CPAB transformers w.r.t. $\boldsymbol{\theta}$ using our CUDA implementation[1] while TensorFlow's auto differentiation (using the chain rule) takes care of the rest. We remark that in comparison with the finite-difference approach (which only approximates $\partial T^{\boldsymbol{\theta}}(\boldsymbol{x})/\partial\boldsymbol{\theta}$), our implementation is not only more accurate but also considerably faster. For details about how the gradient of the CPAB transformations w.r.t. $\boldsymbol{\theta}$ is computed we refer to [1].

[1] As mentioned in the paper, the CPAB derivative, $\partial T^{\boldsymbol{\theta}}(\boldsymbol{x})/\partial\boldsymbol{\theta}$, whose mathematical details can be found in [1], has no closed-form expression. Rather, it is given only via the solution of a system of coupled integral equations [3]. As such, TensorFlows auto-differentiation is inapplicable for computing it.

Finally, in comparison to the implementation from [2, 3], we added two additional parallelism levels; while in [2, 3] parallelization was done only over different pixels, here we also parallelize over different images as well as the d components of the CPAB gradient.

## 2. Tessellation size

In the paper we do not go into details about the hyperparameters $n_x, n_y$, that determine the tessellation's size, and thereby $\dim(\boldsymbol{\theta})$. For completeness, in Table 1 we report an experiment on the LFW(restricted) dataset, where we experimented with different tessellations.

| Tessellation | $1 \times 1$ | $2 \times 2$ | $3 \times 3$ | $4 \times 4$ | $5 \times 5$ |
|---|---|---|---|---|---|
| $d = \dim(\theta)$ | 10 | 20 | 34 | 58 | 90 |
| Time [s] | 3456 | 5691 | 8259 | 13374 | 19733 |
| Accuracy [%] | 82.3 | 86.8 | 89.7 | 89.3 | 88.2 |

**Table 1:** Classification accuracy on the LFW (restricted) dataset for different tessellation sizes.

In retrospect, the results Table 1 show that we could have used a smaller tessellation size, with increase in performance (for this particular dataset) and a decrease in computational time.

| | | $\dim(\boldsymbol{\theta})$ | | | |
|---|---|---|---|---|---|
| | Parameter space | Cont. | Cont. + ZB | Cont. + VP | Cont. + ZB + VP |
| $\mathcal{P}_2$ | 96 | 20 | 14 | 10 | 3 |
| $\mathcal{P}_4$ | 386 | 58 | 32 | 15 | 10 |

**Table 2:** The initial parameter space of two tessellations, $\mathcal{P}_2 = [2, 2]$ and $\mathcal{P}_4 = [4, 4]$, and after enforcing different sets of constrains. Evidently from the table, we achieve a highly-flexible transformation with a low-dimensional $\boldsymbol{\theta}$. Legend: Cont = Continuity, ZB = Zero Boundary, VP = Volume preservation.

Worth nothing about our method, is the ability of incorporating other constrains than continuity into the model. In [2] and its supplementary material Freifeld*et al*. go into much more detail; Table 2 shows the parametrization size $\dim(\boldsymbol{\theta})$ for different choices of tessellation and constrain combinations.

## 3. Analytic matrix exponential for 3x3 matrices

As evident from algorithm 2, our algorithm requires the evaluation of the matrix exponential. Thus, we have here derived the analytical expression for the matrix exponential for matrices with the following special form

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 0 \end{bmatrix}$$

Define $y = a^2 - 2 \cdot a \cdot e + 4 \cdot b \cdot d + e^2$ and $x = \sqrt{y}$. The resulting matrix exponential is given by

$$\mathbf{expm}(A) = \begin{bmatrix} ea & eb & ec \\ ed & ee & ef \\ 0 & 0 & 1 \end{bmatrix}$$

where the entries are determined by the value of $y$:

$$ea = \begin{cases} -(4 \cdot ((a-e) \cdot sinx + cosx \cdot x)) \cdot (a \cdot e - b \cdot d) \cdot expea \cdot nomc & \text{if } y < 0 \\ -2 \cdot (a \cdot e - b \cdot d) \cdot ((a - e - x) \cdot expeam - (a - e + x) \cdot expeap) \cdot nomr & \text{if } y > 0 \\ 2 \cdot (a - e + 2) \cdot (a \cdot e - b \cdot d) \cdot expea/ea2 & \text{if } y = 0 \end{cases}$$

$$eb = \begin{cases} -8 \cdot b \cdot (a \cdot e - b \cdot d) \cdot sinx \cdot expea \cdot nomc & \text{if } y < 0 \\ -4 \cdot b \cdot (expeam - expeap) \cdot (a \cdot e - b \cdot d) \cdot nomr & \text{if } y > 0 \\ 4 \cdot b \cdot (a \cdot e - b \cdot d) \cdot expea / ea2 & \text{if } y = 0 \end{cases}$$

$$ec = \begin{cases} -4 \cdot (((-c \cdot e^2 + (a \cdot c + b \cdot f) \cdot e + b \cdot (a \cdot f - 2 \cdot c \cdot d)) \cdot sinx - ... \\ \quad cosx \cdot x \cdot (b \cdot f - c \cdot e)) \cdot expea + (b \cdot f - c \cdot e) \cdot x) \cdot nomc & \text{if } y < 0 \\ (((4 \cdot c \cdot d - 2 \cdot f \cdot eap) \cdot b - 2 \cdot c \cdot e \cdot (a - e - x)) \cdot expeam + ... \\ \quad ((-4 \cdot c \cdot d + 2 \cdot f \cdot eam) \cdot b + 2 \cdot c \cdot e \cdot (a - e + x)) \cdot expeap + 4 \cdot (b \cdot f - c \cdot e) \cdot x) \cdot nomr & \text{if } y > 0 \\ ((-2 \cdot c \cdot e^2 + (2 \cdot b \cdot f + 2 \cdot c \cdot (a + 2)) \cdot e + 2 \cdot b \cdot (-2 \cdot c \cdot d + f \cdot (a - 2))) \cdot expea + ... \\ 4 \cdot b \cdot f - 4 \cdot c \cdot e) / ea2 & \text{if } y = 0 \end{cases}$$

$$ed = \begin{cases} -8 \cdot d \cdot (a \cdot e - b \cdot d) \cdot sinx \cdot expea \cdot nomc & \text{if } y < 0 \\ -4 \cdot d \cdot (expeam - expeap) \cdot (a \cdot e - b \cdot d) \cdot nomr & \text{if } y > 0 \\ 4 \cdot d \cdot (a \cdot e - b \cdot d) \cdot expea / ea2 & \text{if } y = 0 \end{cases}$$

$$ee = \begin{cases} 4 \cdot ((a - e) \cdot sinx - cosx \cdot x) \cdot (a \cdot e - b \cdot d) \cdot expea \cdot nomc & \text{if } y < 0 \\ 2 \cdot (a \cdot e - b \cdot d) \cdot ((a - e + x) \cdot expeam - (a - e - x) \cdot expeap) \cdot nomr & \text{if } y > 0 \\ -(2 \cdot (a - e - 2)) \cdot (a \cdot e - b \cdot d) \cdot expea / ea2 & \text{if } y = 0 \end{cases}$$

$$ef = \begin{cases} 4 \cdot (((a^2 \cdot f + (-c \cdot d - e \cdot f) \cdot a + d \cdot (2 \cdot b \cdot f - c \cdot e)) \cdot sinx - ... \\ \quad x \cdot cosx \cdot (a \cdot f - c \cdot d)) \cdot expea + x \cdot (a \cdot f - c \cdot d)) \cdot nomc & \text{if } y < 0 \\ ((2 \cdot a^2 \cdot f + (-2 \cdot c \cdot d - 2 \cdot f \cdot (e - x)) \cdot a + 4 \cdot d \cdot (b \cdot f - (1/2) \cdot c \cdot (e + x))) \cdot expeam + ... \\ \quad (-2 \cdot a^2 \cdot f + (2 \cdot c \cdot d + 2 \cdot f \cdot (e + x)) \cdot a - 4 \cdot (b \cdot f - (1/2) \cdot c \cdot (e - x)) \cdot d) \cdot expeap - ... \\ \quad (4 \cdot (a \cdot f - c \cdot d)) \cdot x) \cdot nomr & \text{if } y > 0 \\ ((-2 \cdot a^2 \cdot f + (2 \cdot c \cdot d + 2 \cdot f \cdot (e + 2)) \cdot a - 4 \cdot d \cdot (b \cdot f - 0.5 \cdot c \cdot (e - 2))) \cdot expea - ... \\ \quad 4 \cdot a \cdot f + 4 \cdot c \cdot d) / ea2 & \text{if } y = 0 \end{cases}$$

where:

$$nomc = 1/((-a^2 - 2 \cdot a \cdot e - e^2 - x^2) \cdot x) \qquad nomr = 1/(x \cdot eam \cdot eap)$$
$$sinx = \sin(0.5 \cdot x) \qquad expeap = \exp(0.5 \cdot eap)$$
$$cosx = \cos(0.5 \cdot x) \qquad expeam = \exp(0.5 \cdot eam)$$
$$expea = \exp(0.5 \cdot (a + e)) \qquad ea2 = (a + e)^2$$
$$eap = a + e + x \qquad expea = \exp(0.5 \cdot (a + e))$$
$$eam = a + e - x$$

For a complete deviation of the matrix exponential, please look at the supplied Maple document `matrix_exponential.mw` or respective pdf file `matrix_exponential.pdf`.

## 4. Transformed samples

In Figs. 1–4 we have shown additional transformed samples for the different experiments in the paper. We have not included samples from the homographic and diffeomorphic affine transformer, since these are very similar to the transformations of the affine transformer.

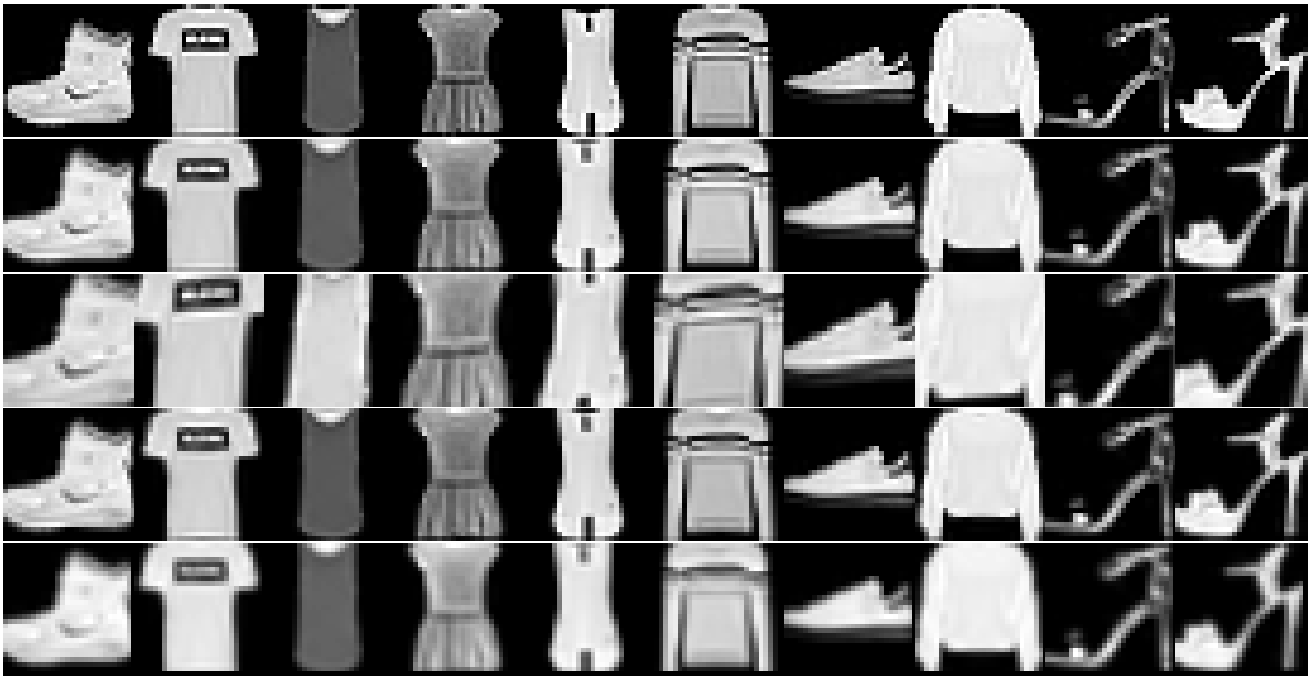

**Figure 1:** Examples of learned transformations for the different models on the fashion dataset. From top to bottom: No transformer, Affine transformations, TPS transformations, CPAB transformations, Affine+CPAB transformations.

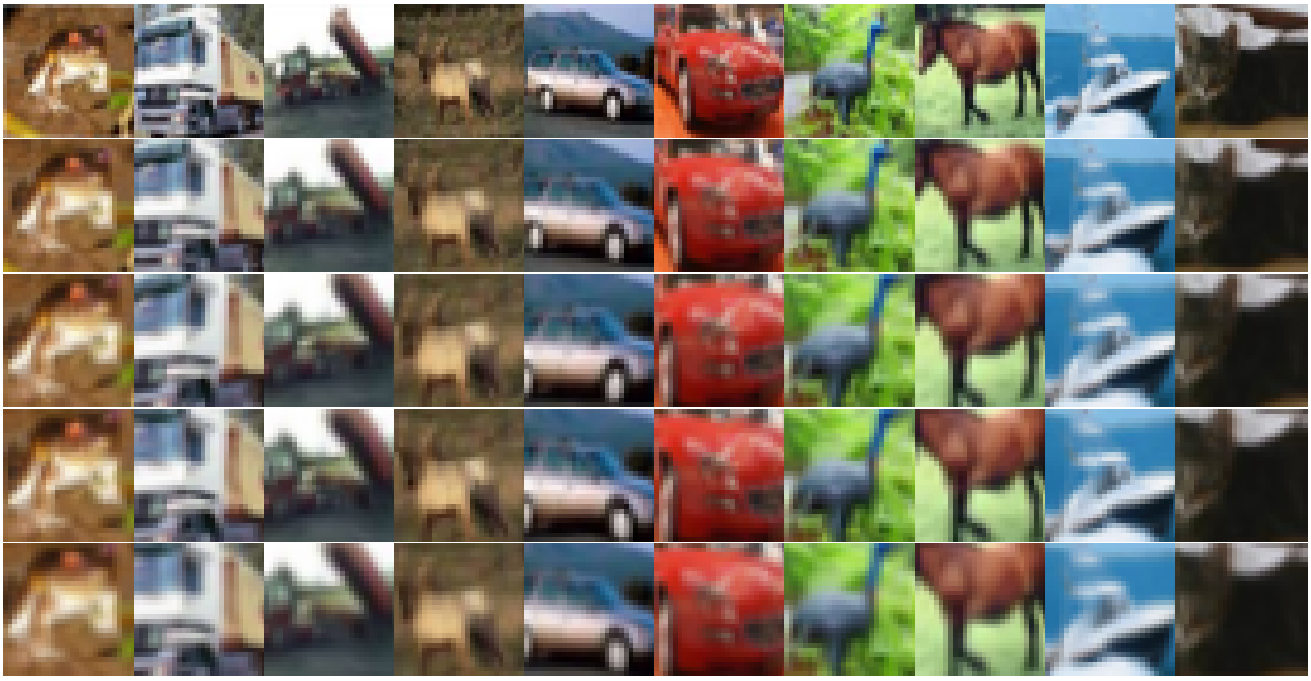

**Figure 2:** Examples of learned transformations for the different models on the CIFAR10 dataset. From top to bottom: No transformer, Affine transformations, TPS transformations, CPAB transformations, Affine+CPAB transformations.

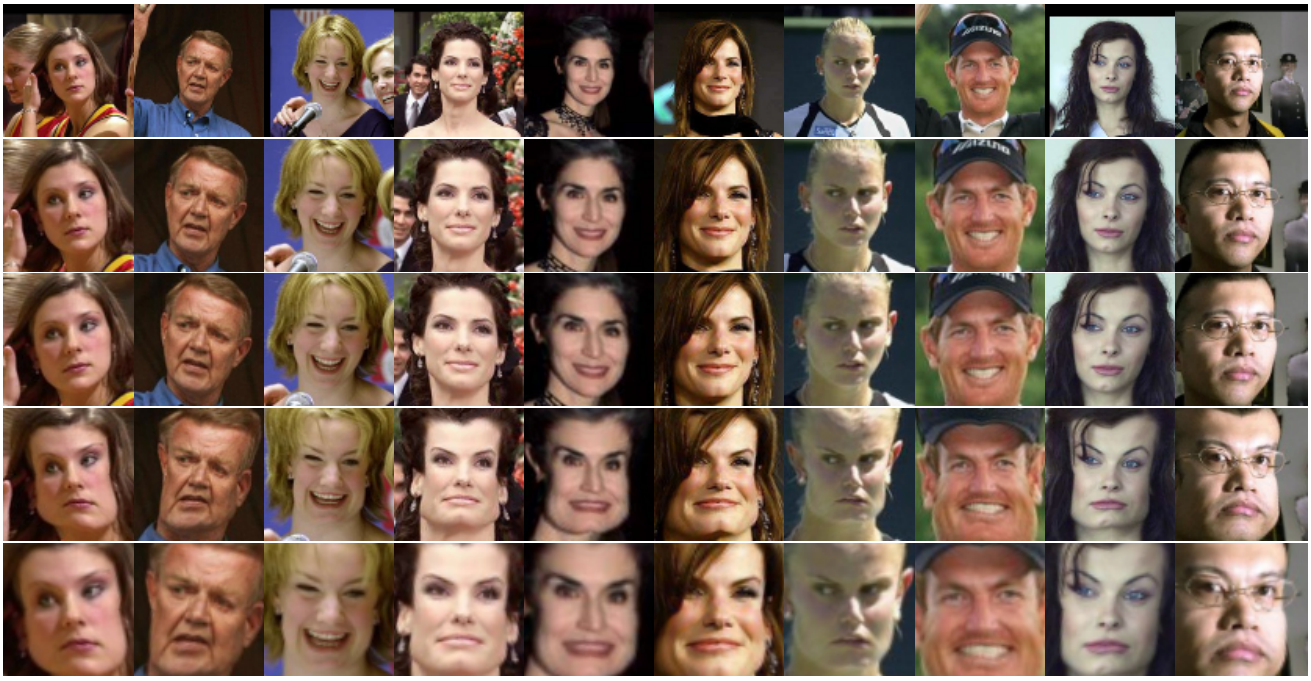

**Figure 3:** Examples of learned transformations for the different models on the LFW dataset. From top to bottom: No transformer, Affine transformations, TPS transformations, CPAB transformations, Affine+CPAB transformations.

**Figure 4:** Examples of learned transformations for the different models on the LFW dataset. From top to bottom: No transformer, Affine transformations, TPS transformations, CPAB transformations, Affine+CPAB transformations.

## References


[1] O. Freifeld. Deriving the CPAB derivative. Technical report, The Department of Computer Science, Ben-Gurion University, 2018. 2

[2] O. Freifeld, S. Hauberg, K. Batmanghelich, and J. W. Fisher. Highly-expressive spaces of well-behaved transformations: Keeping it simple. In *ICCV*, 2015. 1, 2, 3, 4

[3] O. Freifeld, S. Hauberg, K. Batmanghelich, and J. W. Fisher. Transformations based on continuous piecewise-affine velocity fields. *IEEE TPAMI*, 2017. 1, 2, 3