Deep Mixture of Linear Inverse Regressions Applied to Head-Pose Estimation

Stéphane Lathuilière[1], Rémi Juge[1], Pablo Mesejo[1], Rafael Muñoz-Salinas[2], Radu Horaud[1]
[1]Inria Grenoble Rhône-Alpes, France
[2]Computing and Numerical Analysis Department, Cordoba University, Spain
stephane.lathuiliere@inria.fr

Abstract

Convolutional Neural Networks (ConvNets) have become the state-of-the-art for many classification and regression problems in computer vision. When it comes to regression, approaches such as measuring the Euclidean distance of target and predictions are often employed as output layer. In this paper, we propose the coupling of a Gaussian mixture of linear inverse regressions with a ConvNet, and we describe the methodological foundations and the associated algorithm to jointly train the deep network and the regression function. We test our model on the head-pose estimation problem. In this particular problem, we show that inverse regression outperforms regression models currently used by state-of-the-art computer vision methods. Our method does not require the incorporation of additional data, as it is often proposed in the literature, thus it is able to work well on relatively small training datasets. Finally, it outperforms state-of-the-art methods in head-pose estimation using a widely used head-pose dataset. To the best of our knowledge, we are the first to incorporate inverse regression into deep learning for computer vision applications.

1. Introduction

Deep learning has been playing a very important role in the computer vision field during the last years. Many methods have been proposed for challenging tasks, such as image classification [17, 33] or object detection [12, 29]. State-of-the-art results in these classification tasks have been achieved with the use of Convolutional Neural Networks (ConvNets) trained to minimize a loss function on the output of a softmax layer. Besides classification, ConvNets have also been employed to solve regression problems, $e.g.$ image registration [21], organ volume estimation [37], or salient object detection [19], just to name a few. In most cases, when dealing with regression problems, the last softmax layer used in classification tasks is replaced with a fully connected regression layer with linear or sigmoid activations that minimizes an Euclidean loss. This type of configuration ignores the existence of other regression techniques, like inverse regression models, that are suitable in high-dimensional to low-dimensional settings [6, 16, 18, 25] which are of particular interest in computer vision. To identify the benefit of using inverse regression instead of forward (or standard) regression, let's consider the simple case in which we want to estimate a linear regression from $\boldsymbol{x} \in \mathbb{R}^D$ to $y \in \mathbb{R}$, with N training samples such that $D \gg N$. The problem is ill-posed in the case of forward regression ($y = \boldsymbol{a}^\top \boldsymbol{x}, \boldsymbol{a} \in \mathbb{R}^D$) and regularization is required because one needs to estimate D parameters from only N equations. Interestingly, for linear models in the inverse regression setting ($\boldsymbol{x} = \boldsymbol{a}^* y, \boldsymbol{a}^* \in \mathbb{R}^D$), the problem is well defined since one still needs to estimate a set of D parameters but from $D \times N$ equations.

The most common strategy when using deep learning consists in taking an architecture that has already proven to be competitive (in our case VGG-16 [30], the model with the smallest localization error on the ImageNet Large-Scale Visual Recognition Challenge 2014 [28]), download a pre-trained model, slightly modify it ($e.g.$ replacing the last softmax layer with a regression layer), and fine-tune it on the particular application under study. In this scenario, we propose a new output layer designed specifically to perform regression. This output layer is a Gaussian mixture of inverse linear regressions. Mixtures of inverse linear regressions [6] have already been successfully applied to hyperspectral image analysis [5], sound-source localization [7] and head-pose estimation [8]. Moreover, it has been extended to mixtures of t-distributions [25] which provides an inverse regression formulation that is robust to outliers.

We believe that inverse regression models are well-suited in the deep learning framework because deep neural networks represent images in high-dimensional feature spaces that must subsequently be mapped onto low-dimensional manifolds. Interesting enough, recognizing the caveats of high-dimensional regression and exploring inverse regres-

sion models have received little attention in the literature of both computer vision and deep learning fields. In this work, we propose to couple a Gaussian mixture of linear inverse regressions with a ConvNet, we describe the methodological foundations and the associated algorithm to jointly train the network and the regression function, and we evaluate our model on the problem of head-pose estimation. The training algorithm we propose is a fine-tuning procedure designed to transfer the representations learned in a classification task to our specific problem. However, using pre-trained ConvNets to estimate the head-pose is not an easy task because very deep ConvNets have been trained to classify objects independently of their pose. As a consequence, the deep features have been designed to be as pose-invariant as possible. Conversely, in our case we want the model to be highly dependent on the pose but independent of the person.

In this paper, we show that inverse regression outperforms L_2-based regression models currently used in head-pose estimation. Our proposal works well without the use of additional data, in opposition to other approaches yielding state-of-the-art results. Finally, it outperforms state-of-the-art methods in head-pose estimation using a widely used head-pose dataset. The implementation of the proposed method used in our experiments is publicly available[1].

2. Related Work

In this section, we discuss deep learning approaches in conjunction with regression methods. In addition, we review the related work on head-pose estimation, since it is used in our experiments for evaluation and comparison with other methods.

Deep regression algorithms, where the goal is to predict a set of interdependent continuous values, have been well studied in recent years. For instance, in human pose estimation, the target represents the positions of the human-body joints [34]; in head-pose estimation, the target represent the yaw, pitch and roll angles [22]; and in facial landmark detection, the predicted target denotes the image locations of the facial points [32]. In all aforementioned references, a ConvNet has been trained using a loss function that measures the L_2 distance of the prediction from the target, without considering its vulnerability to outliers (as evidenced by [2], where the authors argue that training a ConvNet using a loss function that is robust to outliers, *e.g.* Tukey's biweight function, results in faster convergence and better generalization). Also, when substituting the final regression layer by a more sophisticated regression, most of previous networks employed discriminative approaches like random forests [37] or support vector regression [11]. Finally, the vast majority of existing works tries to solve a specific computer vision task which is expressed as a regression problem without focusing onto the regression framework itself. To the best of our knowledge, there are no other methods that attempt to incorporate inverse regression into deep learning for computer vision applications.

Head-pose estimation is an important cue for tasks such as human-robot interaction, computer-human interaction, analysis of human behavior, or driver-assistance systems [23]. The pose is typically expressed by three angles that describe the orientation of the head (looking up or down: pitch, left or right: yaw, and tilting left or right: roll). The estimation of the pose parameters is challenging due to changing illumination conditions, to the background scene, to partial occlusions, and to inter-person and intra-person variabilities.

There are few recent papers using deep learning to regress the angles that determine the human head pose. The pioneering work of Osadchy et al. [24] synergistically performs face detection and pose estimation by employing a ConvNet to map face images to points on a manifold parameterized by pose, and non-face images to points away from that manifold. In [22], the authors use GoogLeNet [33] and replace the last softmax layer with an Euclidean loss layer that measures the L_2 distance of the prediction from the target. Liu et al. [20] train on synthetic head images and employ a quite simple ConvNet (3 convolutional and 2 fully connected layers; with a linear activation function to predict the head poses in the output layer) to perform head-pose estimation. A quite similar approach can be found in [1], [36] and [27], where slightly different ConvNet architectures are used. Finally, HyperFace [26] is a single ConvNet model (5 convolutional layers along with 3 fully connected layers using the Euclidean loss to train the head-pose estimates) for simultaneous face detection, landmark localization, pose estimation and gender classification. The proposed method is one of the first attempts to use a very deep pre-trained network to effectively tackle the head-pose estimation problem.

3. Mixture of Linear Inverse Regressions

In this section we describe in detail the regression layer of the proposed model. We consider a deep neural network $\boldsymbol{\phi}$ with weights $\boldsymbol{w}$ that maps an image $\boldsymbol{i} \in \mathbb{R}^M$ onto a *high-dimensional* feature vector $\boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{i}; \boldsymbol{w}) \in \mathbb{R}^D$. The regression $\boldsymbol{r}^*$, with parameters $\boldsymbol{\theta}^*$, maps $\boldsymbol{x}$ onto a *low-dimensional* target $\boldsymbol{y} = \boldsymbol{r}^*(\boldsymbol{x}; \boldsymbol{\theta}^*) \in \mathbb{R}^L$ with $L \ll D$. The regression layer can be expressed probabilistically in the following way. Let $\boldsymbol{i}$, $\boldsymbol{x}$ and $\boldsymbol{y}$ be realizations of the random variables $\boldsymbol{I}$, $\boldsymbol{X}$ and $\boldsymbol{Y}$. The goal is to estimate the target $\boldsymbol{Y}$ given an input image $\boldsymbol{I}$ and the model parame-

[1] https://team.inria.fr/perception/research/dmlir/

ters $(\boldsymbol{w}, \boldsymbol{\theta}^*)$, i.e. the conditional density $p(\boldsymbol{Y}|\boldsymbol{\phi}(\boldsymbol{I}; \boldsymbol{w}); \boldsymbol{\theta}^*)$. Once this posterior distribution is estimated, one can predict the target corresponding to an input based on the conditional expectation of the target, namely $\hat{\boldsymbol{y}} = \boldsymbol{r}^*(\boldsymbol{\phi}(\boldsymbol{i}; \bar{\boldsymbol{w}}); \boldsymbol{\theta}^*) = \mathbb{E}[\boldsymbol{y}|\boldsymbol{\phi}(\boldsymbol{i}; \bar{\boldsymbol{w}}); \boldsymbol{\theta}^*]$, where $\bar{\boldsymbol{w}}$ denotes the optimized values of the weights.

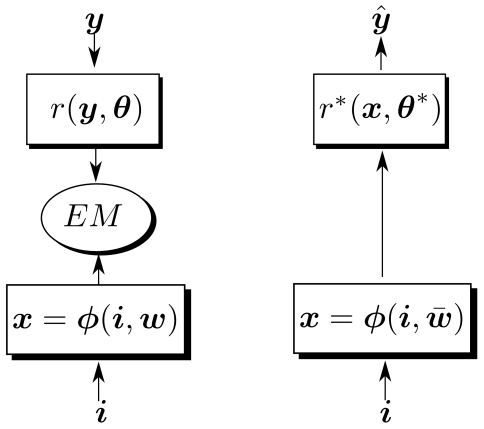
Estimating a regression function defined over a high-dimensional space, as above, is generally difficult because one has to estimate a large number of parameters, typically of the order of D^2. We bypass this difficulty by training an *inverse regression*, *e.g.* Fig. 1. More precisely, at training the low-dimensional target $\boldsymbol{Y}$ is the input to the regressor, while the high-dimensional feature vector $\boldsymbol{X}$ is the output. Hence, $\boldsymbol{Y}$ is assumed to lie on a low-dimensional (linear or non-linear) manifold embedded in $\mathbb{R}^D$ and parameterized by $\boldsymbol{X}$. Choosing the low-dimensional variable to be the input implies a smaller number of parameters, typically $L(D + L)$), to be estimated. Hence, the parameters of the *inverse* conditional density are estimated at training, i.e. $p(\boldsymbol{\phi}(\boldsymbol{I}; \boldsymbol{w})|\boldsymbol{Y}; \boldsymbol{\theta})$, from which the *forward* conditional density is then derived and used for prediction, i.e. $p(\boldsymbol{Y}|\boldsymbol{\phi}(\boldsymbol{I}; \bar{\boldsymbol{w}}); \boldsymbol{\theta}^*)$. Such an inverse regression can be implemented with either non-parametric [18] or parametric [6, 25] methods. The advantage of the latter over the former is twofold: (i) the inverse parameters $\boldsymbol{\theta}$ can be estimated in closed-form either with Gaussian mixtures [6] or with mixtures of t-distributions [25], and (ii) the forward parameters $\boldsymbol{\theta}^*$ can be analytically derived from the inverse parameters with both these two mixture models. Moreover, a parametric model allows us to alternate between the optimization of the network weights and of the regression parameters.

We consider the following mixture of K affine regressions:

$$\boldsymbol{X} = \sum_{k=1}^{K} \mathbb{I}(Z = k)(\mathbf{A}_k \boldsymbol{Y} + \boldsymbol{b}_k + \boldsymbol{E}_k), \tag{1}$$

where $\mathbb{I}$ is the indicator function, Z is a hidden variable such that $Z = k$ if and only if $\boldsymbol{X}$ is the result of mapping $\boldsymbol{Y}$ using the affine transformation $\mathbf{A}_k \boldsymbol{Y} + \boldsymbol{b}_k$, with $\mathbf{A}_k \in \mathbb{R}^{D \times L}$ and $\boldsymbol{b}_k \in \mathbb{R}^D$, and $\boldsymbol{E}_k \in \mathbb{R}^D$ is an error vector. By marginalization over Z, the joint probability of $\boldsymbol{y}$ and $\boldsymbol{x}$ can be written as $p(\boldsymbol{x}, \boldsymbol{y}; \boldsymbol{\theta}, \boldsymbol{w}) = \sum_{k=1}^{K} p(\boldsymbol{x}|\boldsymbol{y}, Z = k; \boldsymbol{\theta}, \boldsymbol{w}) p(\boldsymbol{y}|Z = k; \boldsymbol{\theta}) p(Z = k; \boldsymbol{\theta})$. Under the assumption that $\boldsymbol{E}_k$ is a zero-mean Gaussian variable with diagonal covariance $\boldsymbol{\Sigma}_k \in \mathbb{R}^{D \times D}$, we obtain that $p(\boldsymbol{x}|\boldsymbol{y}, Z = k; \boldsymbol{\theta}, \boldsymbol{w}) = \mathcal{N}(\boldsymbol{x}; \mathbf{A}_k \boldsymbol{y} + \boldsymbol{b}_k, \boldsymbol{\Sigma}_k)$. We further assume that $\boldsymbol{Y}$ follows a mixture of Gaussians. We can now write $p(\boldsymbol{y}|Z = k; \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{c}_k, \boldsymbol{\Gamma}_k)$ and $p(Z = k; \boldsymbol{\theta}) = \pi_k$, where $\boldsymbol{c}_k \in \mathbb{R}^L$, $\boldsymbol{\Gamma}_k \in \mathbb{R}^{L \times L}$ and $\sum_{k=1}^{K} \pi_k = 1$.

Altogether, the regression layer is described by the parameter set $\boldsymbol{\theta} = \{\boldsymbol{c}_k, \boldsymbol{\Gamma}_k, \pi_k, \mathbf{A}_k, \boldsymbol{b}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$. Both $\boldsymbol{\theta}$ and



(a) Inverse training (b) Forward prediction

Figure 1: The method proposed in this paper performs training by gluing inverse regression ($\boldsymbol{r}$ parameterized by $\boldsymbol{\theta}$) and network fine-tuning ($\boldsymbol{\phi}$ parameterized by $\boldsymbol{w}$) in an EM procedure. The parameters $\boldsymbol{\theta}^*$ of the forward regression $\boldsymbol{r}^*$ can be derived analytically from $\boldsymbol{\theta}$, which allows to predict a target $\boldsymbol{y}$ associated with an input $\boldsymbol{i}$.

$\boldsymbol{w}$ can be estimated via the EM algorithm described in detail in Sec. 4. Once optimal values for $\boldsymbol{\theta}$ and $\boldsymbol{w}$ are estimated, namely $\bar{\boldsymbol{\theta}}$ and $\bar{\boldsymbol{w}}$, the inverse conditional density can be written as:

$$p(\boldsymbol{\phi}(\boldsymbol{i}; \bar{\boldsymbol{w}})|\boldsymbol{y}; \bar{\boldsymbol{\theta}}) = \sum_{k=1}^{K} \bar{\nu}_k \mathcal{N}(\boldsymbol{\phi}(\boldsymbol{i}; \bar{\boldsymbol{w}}); \bar{\mathbf{A}}_k \boldsymbol{y} + \bar{\boldsymbol{b}}_k, \bar{\boldsymbol{\Sigma}}_k), \tag{2}$$

with $\bar{\nu}_k = \bar{\pi}_k \mathcal{N}(\boldsymbol{y}; \bar{\boldsymbol{c}}_k, \bar{\boldsymbol{\Gamma}}_k) / \sum_{j=1}^{K} \bar{\pi}_j \mathcal{N}(\boldsymbol{y}; \bar{\boldsymbol{c}}_j \bar{\boldsymbol{\Gamma}}_j)$. The forward predictive distribution can then be expressed as:

$$p(\boldsymbol{y}|\boldsymbol{\phi}(\boldsymbol{i}; \bar{\boldsymbol{w}}); \boldsymbol{\theta}^*) = \sum_{k=1}^{K} \nu_k^* \mathcal{N}(\boldsymbol{y}; \mathbf{A}_k^* \boldsymbol{\phi}(\boldsymbol{i}; \bar{\boldsymbol{w}}) + \boldsymbol{b}_k^*, \boldsymbol{\Sigma}_k^*), \tag{3}$$

with $\nu_k^* = \pi_k^* \mathcal{N}(\boldsymbol{x}; \boldsymbol{c}_k^*, \boldsymbol{\Gamma}_k^*) / \sum_{j=1}^{K} \pi_j^* \mathcal{N}(\boldsymbol{x}; \boldsymbol{c}_j^* . \boldsymbol{\Gamma}_j^*)$ and with parameters $\boldsymbol{\theta}^* = \{\boldsymbol{c}_k^*, \boldsymbol{\Gamma}_k^*, \pi_k^*, \mathbf{A}_k^*, \boldsymbol{b}_k^*, \boldsymbol{\Sigma}_k^*\}_{k=1}^{K}$. An interesting feature of this model is that the forward parameters $\boldsymbol{\theta}^*$ can be expressed analytically from the inverse parameters $\boldsymbol{\theta}$:

$$\begin{aligned}
\boldsymbol{c}_k^* &= \mathbf{A}_k \boldsymbol{c}_k + \boldsymbol{b}_k, \\
\boldsymbol{\Gamma}_k^* &= \boldsymbol{\Sigma}_k + \mathbf{A}_k \boldsymbol{\Gamma}_k \mathbf{A}_k^\top, \\
\pi_k^* &= \pi_k, \\
\mathbf{A}_k^* &= \boldsymbol{\Sigma}_k^* \mathbf{A}_k^\top \boldsymbol{\Sigma}_k^{-1}, \\
\boldsymbol{b}_k^* &= \boldsymbol{\Sigma}_k^* (\boldsymbol{\Gamma}_k^{-1} \boldsymbol{c}_k - \mathbf{A}_k^\top \boldsymbol{\Sigma}_k^{-1} \boldsymbol{b}_k), \\
\boldsymbol{\Sigma}_k^* &= (\boldsymbol{\Gamma}_k^{-1} + \mathbf{A}_k^\top \boldsymbol{\Sigma}_k^{-1} \mathbf{A}_k)^{-1}.
\end{aligned}$$

As a consequence, one can use the conditional expectation associated with (3) to predict a target:

$$\hat{\boldsymbol{y}} = \boldsymbol{r}^*(\boldsymbol{\phi}(\boldsymbol{i}; \bar{\boldsymbol{w}}); \boldsymbol{\theta}^*) = \sum_{k=1}^{K} \nu_k^* (\mathbf{A}_k^* \boldsymbol{\phi}(\boldsymbol{i}; \bar{\boldsymbol{w}}) + \boldsymbol{b}_k^*). \quad (4)$$

4. Training the Proposed Model

In this section we describe the estimation of the model parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$ based on an expectation-maximization algorithm and using a training dataset $\{\boldsymbol{i}_n, \boldsymbol{y}_n\}_{n=1}^N$, i.e. Alg. 1. Fig. 2 shows the proposed training applied to a toy example.

The E-step updates the posterior probabilities with the following expression:

$$\begin{aligned} \mu_{nk}^{(i+1)} =& p(Z_n = k | \boldsymbol{y}_n, \boldsymbol{x}_n; \boldsymbol{\theta}^{(i)}, \boldsymbol{w}^{(i)}) \\ =& \frac{\pi_k^{(i)} p(\boldsymbol{y}_n, \boldsymbol{x}_n | Z_n = k; \boldsymbol{\theta}^{(i)}, \boldsymbol{w}^{(i)})}{\sum_{j=1}^{K} \pi_j^{(i)} p(\boldsymbol{y}_n, \boldsymbol{x}_n | Z_n = j; \boldsymbol{\theta}^{(i)}, \boldsymbol{w}^{(i)})} \end{aligned} \quad (5)$$

with:

$$\begin{aligned} p(\boldsymbol{y}_n, \boldsymbol{x}_n | Z_n = k; \boldsymbol{\theta}, \boldsymbol{w}) =& p(\boldsymbol{x}_n | \boldsymbol{y}_n, Z_n = k; \boldsymbol{\theta}, \boldsymbol{w}) \\ & \times p(\boldsymbol{y}_n | Z_n = k; \boldsymbol{\theta}) \end{aligned} \quad (6)$$

The M-step performs the following maximization:

$$\begin{aligned} &(\boldsymbol{\theta}^{(i+1)}, \boldsymbol{w}^{(i+1)}) = \\ &\underset{(\boldsymbol{\theta}, \boldsymbol{w})}{\text{argmax}}\, \mathbb{E}\big[\log\, p((\boldsymbol{x}, \boldsymbol{y}, Z)_{1:N}; \boldsymbol{\theta}, \boldsymbol{w}) | (\boldsymbol{y}, \boldsymbol{i})_{1:N}; \boldsymbol{\theta}^{(i)}, \boldsymbol{w}^{(i)}\big] \end{aligned} \quad (7)$$

This is further decomposed in three sub-steps: M-GMM-step, M-Mapping-step, and the M-Network-step, i.e. Alg. 1. The update formulae for the parameters $\boldsymbol{\theta}$ can be found in [6]. The network's weights are estimated as follows. By developing the expected complete-data log-likelihood (7) and after keeping the terms that depend on $\boldsymbol{w}$, we obtain the following loss function:

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}) =& \sum_{n=1}^{N} \sum_{k=1}^{K} p(Z_n = k | (\boldsymbol{y}_n, \boldsymbol{i}_n)) \\ & \times \log\, p(\boldsymbol{\phi}(\boldsymbol{i}_n, \boldsymbol{w}) | \boldsymbol{y}_n, Z_n = k; \boldsymbol{\theta}) \\ =& \sum_{n=1}^{N} \sum_{k=1}^{K} \mu_{nk} \log\, p(\boldsymbol{\phi}(\boldsymbol{i}_n, \boldsymbol{w}) | \boldsymbol{y}_n, Z_n = k; \boldsymbol{\theta}) \\ =& \sum_{n=1}^{N} \sum_{k=1}^{K} \mu_{nk} \log\, \mathcal{N}(\boldsymbol{\phi}(\boldsymbol{i}_n, \boldsymbol{w}); \mathbf{A}_k \boldsymbol{y}_n + \boldsymbol{b}_k, \boldsymbol{\Sigma}_k) \end{aligned} \quad (8)$$

Data: Training dataset $(\boldsymbol{i}, \boldsymbol{y})_{n=1:N}^N$, number of components K, and convergence threshold $\epsilon \in \mathbb{R}$;
Result: $\boldsymbol{\theta}$ and $\boldsymbol{w}$;
Initialize $\boldsymbol{\theta}^{(0)}$ and $\boldsymbol{w}^{(0)}$;
while $||\boldsymbol{\theta}^{(i+1)}$-$\boldsymbol{\theta}^{(i)}|| > \epsilon$ **do**
  E-step: Update the posteriors $\boldsymbol{\mu}^{(i+1)} = \{\mu_{nk}^{(i+1)}\}_{n=1,k=1}^{N,K}$ given the current parameters $\boldsymbol{\theta}^{(i)}$ and weights $\boldsymbol{w}^{(i)}$.
  M-GMM-step: Update the mixture parameters $\{\boldsymbol{c}_k^{(i+1)}, \boldsymbol{\Gamma}_k^{(i+1)}, \pi_k^{(i+1)}, \}_{k=1}^K$ given the posteriors $\boldsymbol{\mu}^{(i+1)}$ and the current mapping parameters and the current network weights;
  M-Mapping-step: Update the affine parameters $\{\mathbf{A}_k^{(i+1)}, \boldsymbol{b}_k^{(i+1)}, \boldsymbol{\Sigma}_k^{(i+1)}\}_{k=1}^K$ given the posteriors $\boldsymbol{\mu}^{(i+1)}$, the mixture parameters and the current network weights, and
  M-Network-step: Update the weights $\boldsymbol{w}^{(i+1)}$ given the posteriors $\boldsymbol{\mu}^{(i+1)}$ and the current parameter values $\boldsymbol{\theta}^{(i+1)}$.
end

Algorithm 1: EM algorithm for deep inverse regression.

If we further assume that the error covariances are isotropic, i.e. $\boldsymbol{\Sigma}_k = \lambda_k^{-1}\mathbb{I}$ where $\lambda_k \in \mathbb{R} > 0$ is the precision associated with each affine transformation, we obtain the following loss function:

$$\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} \sum_{k=1}^{K} \mu_{nk} \lambda_k ||\mathbf{A}_k \boldsymbol{y}_n + \boldsymbol{b}_k - \boldsymbol{\phi}(\boldsymbol{i}_n, \boldsymbol{w})||_2^2 \quad (9)$$

This loss function has the form of a weighted mean-squared error and hence gradient descent techniques for deep neural network optimization are well suited and can be easily used [13]. Notice however that gradient stability issues are common. In particular deep regression can be difficult to train when the target space is unbounded because it is likely to lead to exploding gradient problems [3]. As a consequence, the targets $\boldsymbol{x}_n$ may reach really high values after a few EM iterations. To avoid this problem we use a normalization layer [14]. Moreover this layer avoids converging to the undesirable solution where $\mathbf{A}_k = 0$ and $\boldsymbol{b}_k = 0$ that would maximize the likelihood.

The proposed EM algorithm is initialized as follows. We first perform clustering in the target space using a standard procedure, i.e. K-means with random initializations followed by fitting a GMM. This yields initial values for the posteriors, namely $\mu_{nk}^{(0)}$. Notice however from (6) that the posteriors depend on feature clustering as well and this clustering is not reliable at the start of the algorithm. For this reason, we freeze the E-step (the posteriors are set to their

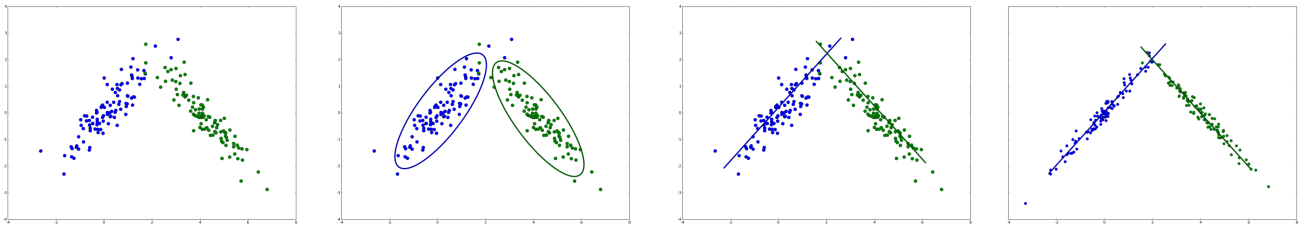


Figure 2: Training the deep inverse regression EM with a toy example, *e.g.* $L = 1$, $D = 2$ and $K = 2$. The E-step computes the posteriors $\mu_{nk}^{(i+1)}$. The M-GMM-step fits a mixture to the data given the posteriors. The M-mapping-step estimates the parameters of the affine regressions (the two lines illustrate the projection of the target space onto the feature space). Finally, the M-network-step fine-tunes the network weights via minimization of a mean-square error loss function.

initial values) and perform a few iterations of the M steps, which amounts to alternate between updating the regression parameters and tuning the network weights.

Finally, it is worth mentioning that the proposed inverse regression can be used with a single Gaussian distribution, i.e. $K = 1$. In this case (1) reduces to $\boldsymbol{X} = \mathbf{A}\boldsymbol{Y} + \boldsymbol{b} + \boldsymbol{E}$ where, again, $\boldsymbol{E}$ is a zero-mean Gaussian variable with diagonal covariance $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times D}$, hence $p(\boldsymbol{y}|\boldsymbol{x};\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{y}|\mathbf{A}\boldsymbol{x} + \boldsymbol{b}, \boldsymbol{\Sigma})$. Notice that it is still interesting to train a low-dimensional to high-dimensional mapping (inverse regression), on the following grounds. The low-to-high regression that we propose to train provides D linear constraints with $D\times(L+2)$ free parameters. Hence, one needs a minimum $L + 2$ image-target training pairs to estimate the model parameters. In contrast, high-to-low regression would have provided L linear constraints for training, with $L\times(D+2)$ free parameters; $D+2$ image-target pairs would have been at least necessary for training.

Because there is no assignment variable in the case of a single Gaussian, the training procedure alternates between estimating the regression parameters $\mathbf{A}$, $\boldsymbol{b}$ and $\boldsymbol{\Sigma}$, and updating the network weights $\boldsymbol{w}$, i.e. M-step iterations of Alg. 1.

5. Experiments

In this section we first describe the datasets used to evaluate the performance of our model. After that, we present the ConvNet architecture employed and the results obtained.

Datasets. The *Biwi Kinect head-pose dataset* [10] consists of over $15K$ images including video recordings of 20 people (16 men, 4 women, some of them recorded twice) using a Kinect camera. During the recordings, the participants freely move their head and the corresponding head angles lie in the intervals $[-60°, 60°]$ (pitch), $[-75°, 75°]$ (yaw), and $[-20°, 20°]$ (roll). Fig. 4 shows examples of the synthetic images generated.

Figure 3: Example frames of the *Biwi head-pose dataset*.

We employed the following protocol to create a fair data partition: we run Support Vector Regression (SVR) [31] on HOG features [4] using an 8-fold cross-validation (21 randomly selected videos for training and the remaining 3 videos for test). After that, we ordered the performance on each fold in terms of their MSE and, finally, we kept the best performing fold for the HOG-based methods and the median performing fold for the VGG-based methods. We acted in this manner to give some advantage to the most simple approaches and to avoid a bias towards our deep learning proposal. In other words, HOG-based methods are trained and tested on the most advantageous fold for them. It is important to notice that we used 20% of the training set as validation set, and that no person appears both in training and test sets.

The main drawback of the Biwi dataset is that most of faces are looking squarely or present small angles. So, the distribution of the targets is almost Gaussian. Since we suspected this property would favor models with a low number of Gaussians, we evaluated our proposal with different target distributions. To do so, we created a synthetic dataset utilizing the MakeHuman 3D software[2] to generate 50 different body models. Parameters like age, gender or color skin were randomly selected by the software. Then, we randomly generated $100K$ images of the models'

[2] www.makehuman.org

head with uniformly distributed angles. To ensure robustness during training, each generated image has a randomly selected lighting position and color for the OpenGL engine lighting system. Fig. 4 shows examples of the synthetic images generated.

Figure 4: Example frames of our synthetic head-pose dataset.

We generated two smaller datasets (approximately $20K$ images per dataset) from this uniformly sampled dataset. First, we selected the images in order to obtain a mixture of two Gaussians centered around $(35°, 60°, 0°)$ and $(-35°, -60°, 0°)$. This dataset is referred to as *S2G dataset (Synthetic with 2 Gaussians)*, and is used to study the impact of the target distribution on the number of Gaussians employed (K). Secondly, we removed some images from the uniformly distributed dataset with a Gaussian of mean $(0°, 0°, 0°)$. This dataset is referred to as *SSV dataset (Synthetic with only Side Views)*. Since the distribution of poses in the SSV dataset is not directly obtained by combining Gaussian distributions, it can be considered as a difficult case for our model in which we could think that K=1 would not necessarily perform well.

ConvNet architectures. In practice, it is relatively difficult to train an entire ConvNet from scratch because it requires a sufficiently large dataset. Furthermore, if the network is very deep, an important amount of computational power would be necessary. A common alternative approach consists in taking a network already trained on ImageNet and use its weights as initialization to train your own ConvNet. In this paper, we use VGG-16 [30]. However, since these networks have been trained to solve a classification task, they have learned to be invariant to the pose of objects. On the contrary, we want our model to be independent of the object but highly dependent on the pose. To face this problem, we use the initialization procedure explained in the last paragraph of Section 4.

The size of the last fully-connected layer of VGG-16 pre-trained on ImageNet is relatively large (4096) as it is designed to recognize approximately 1000 objects. However, in our case we predict only three angles. Thus, we can reduce this dimension in order to reduce the number of parameters in the network and hence the computation burden. To do so, we add a fully connected layer of size 512 with a linear activation function and initialize it with the eigenvectors of a PCA trained on the output of the network. This layer is added before the batch normalizer. This solution presents the advantage that back propagation can be easily performed through this layer.

In practice, we do not exactly iterate between E and M steps as described in Alg.1. We first alternate between the E step and the two M-GMM and M-Mapping steps. When convergence has been reached, we apply the M-Network step. This procedure has two advantages. First, the network weights are not modified before having good mapping functions. Secondly, the $\boldsymbol{\theta}$ updates are performed with the CPU whereas the M-Network utilizes the GPU. The computation is faster if we do not alternate too often between CPU and GPU operations.

Comparison Between Regression Models. We show the results from a series of experiments in order to illustrate the effectiveness of the proposed model. In Table 1, we compare 7 different regression models:

- *$(0°, 0°, 0°)$:* In order to have a reference about the learning ability of our model we introduce this mean pose estimator, i.e. a fictitious method that always return $(0°, 0°, 0°)$ as predicted angles.
- *HOG-SVR:* An SVR is trained using the HOG representation of the input images. The HOG features used in this paper were extracted following the same strategy as in [9], i.e. a HOG pyramid (p-HOG) by stacking HOG descriptors at multiple resolutions and providing a feature vector of dimension 1888.
- *HOG-IR:* An inverse regression (IR) approach equivalent to the model described in [6] with a mixture of 50 affine mappings is trained using HOG features. In other words, *HOG-IR* is equivalent to our proposal in the case K=50, without the M-network step, and using HOG features instead of a deep network.
- *VGG-SVR:* We remove the softmax layer of VGG-16 trained on ImageNet, and we train an SVR to predict the head pose angles from the network features.
- *VGG-IR:* In this case, after removing the softmax layer of the pre-trained VGG-16, our inverse regressor is trained on the output of this network without performing the M-network step and K=50.
- *VGG-FCL-FT:* We replace the softmax layer of a pre-trained VGG-16 by a fully connected layer (FCL) of 3 units and a linear activation function. This layer is trained with a loss function that measures the L_2 distance of the prediction from the target. To draw a fair

comparison, we trained this network with different optimizers and kept the best result. We obtained this result with SGD optimizer, a learning rate of 10^{-3} and a learning decay of 0.5 every 3 epochs. We fine-tune (FT) for 3 epochs only the last layer and then the last four layers. The loss is similar to the commonly employed in the literature [15, 22, 32, 34]. It is important to mention that the results reported here are obtained with the use of a batch normalizer before the regression layer as we obtained poor results without it.

- *VGG-IR-FT:* This is our proposal. The results displayed in Table 1 correspond to the best performing number of Gaussians (K=2), as shown in Table 2.

	Pitch	Yaw	Roll	Mean
$(0°, 0°, 0°)$	23.92	28.50	8.48	20.30
HOG-SVR	8.50	6.45	5.04	6.66
HOG-IR	6.39	5.69	4.77	5.62
VGG-SVR	10.60	19.34	7.79	12.57
VGG-IR	15.26	26.79	10.76	17.60
VGG-FCL-FT	5.65	4.44	**2.93**	4.34
VGG-IR-FT (proposed)	**4.68**	**3.12**	3.07	**3.62**

Table 1: Comparison of different methods on the Biwi head-pose database. Mean absolute errors are given in degrees. The best results are highlighted in bold.

First, we notice that, with HOG features, the inverse regressor outperforms SVR (a forward regressor). However, the results obtained using the deep features given by the pre-trained VGG (*VGG-SVR* and *VGG-IR*) are somewhat disappointing, since the error obtained is much worse than the one obtained by HOG-based methods. This could illustrate the previously mentioned pose-invariance property of the pre-trained VGG-16. These results also show that deep features can hardly be used for head-pose estimation without fine-tuning. In fact, if we fine-tune the network we obtain comparable results to state-of-the-art. The proposed method improves the result by 0.71 degrees with respect to *VGG-FCL-FT*.

In Table 2, we study how the performance of our proposal (*VGG-IR-FT*) evolves as we increase K. We also study how this performance is affected by the type of dataset employed. From the results obtained one can hardly draw a definitive conclusion. On real data, K=2 provides the best performance. However, on cases specifically designed to make fail the model employing K=1 (*S2G* and *SSV*), we obtain comparable results independently of the value of K. This behavior can be explained by the following reasons. First, the difficulty of the synthetic datasets is not enough to make fail the model with K=1. Second, as we can see in (5), the updates of the EM algorithm do not depend only on the target distribution but also rely on the existing clusters in feature space. So, the optimal K cannot be established only by looking at the target distribution. In particular, in *S2G* the model seems to favor the pre-existing clustering in feature space over the mixture of two Gaussians in the target space (as would happen if the best performing model was K=2). In practical terms, one reasonable solution would be to use K=1 by default, since it reduces the complexity of the approach and at the same time provides sufficiently good results. It confirms that deep neural networks are effective linearizers and therefore adding a nonlinearity in the regression layer does not help very much. The benefit of the proposed model comes mainly from the inverse formulation.

	K=1	K=2
Biwi	5.12/3.45/3.39	**4.68**/**3.12**/**3.07**
S2G	2.87/3.54/**1.65**	**2.53**/3.36/1.97
SSV	3.12/**3.75**/2.01	2.86/3.83/2.04
	K=5	K=10
Biwi	5.55/3.74/4.01	5.93/3.45/3.99
S2G	2.63/**3.33**/1.94	2.89/3.35/2.06
SSV	2.77/4.12/**1.95**	**2.74**/3.91/2.18

Table 2: Mean absolute error in degrees for the Pitch/Yaw/Roll angles on three datasets using *VGG-IR-FT* and different K values. The best results per angle and dataset are highlighted in bold.

Head-Pose Estimation State-of-the-Art Comparison. We compare the performance of our proposal with state-of-the-art methods on head-pose estimation.

	Pitch	Yaw	Roll	Mean
Methods using only RGB				
Liu et al.[20]	6.1	6.0	5.7	5.94
Mukherjee et al.[22]	5.18	5.67	/	5.43
Drouard et al.[9]	5.43	4.24	4.13	4.60
VGG-IR-FT (proposed)	**4.68**	**3.12**	**3.07**	**3.62**
Methods using additional information				
*Wang et al.[35]***	8.5	8.8	7.4	8.23
*Mukherjee et al.[22]***	4.76	5.32	/	5.04
*Fanelli et al.[10]***	**3.8**	**3.5**	5.4	4.23
*Liu et al.[20]**	4.5	4.3	**2.4**	**3.73**

Table 3: Comparison of different methods on the Biwi head-pose database. Mean absolute errors are given in degrees. The last four methods use additional or slightly different data (*extra annotation used for training, **3D depth data used). The best results are highlighted in bold.

As can be seen in Table 3, our proposal *VGG-IR-FT* out-

performs state-of-the-art approaches. Drouard et al.[9], in an extension of their previous work [8], employ 4 partially-latent variables in their mixture of linear regression approach to establish the state-of-the-art. Moreover, we can even compete with methods using additional information (see the last four methods in Table 3). Deep learning is not used neither in [10] nor in [35], and both use depth information. Importantly, our approach provides a competitive performance even in absence of this additional information. In [20], Liu et al. train using synthetic data because otherwise they get results comparable to the ones provided by HOG-based methods. Their experience confirms the intuition that a forward regression technique does not perform well without using a large dataset. Among all competitor methods the only one using a very deep network is Mukherjee et al. [22]. They use a GoogLeNet architecture on both RGB and depth images. The superiority of *VGG-IR-FT* can indicate again the benefits of using inverse regression. Finally, an important remark is that all these results could even be further improved by including temporal information, $e.g.$ the temporally stable head-pose estimation proposed by [1].

In order to compare the sensibility to the training set size, we randomly down-sample the Biwi database training set. We show the results in Table 4. We can notice that even employing only 40% of data we are competitive with most of the methods in Table 3. The performance seems to scale linearly with respect to the available training set size. This trend seems to suggest that additional data would further improve the performance.

	Pitch	Yaw	Roll	Mean
20%	9.36	7.00	6.19	7.55
40%	6.2	5.00	5.14	5.45
60%	6.33	4.33	4.18	4.95
80%	5.49	3.77	4.12	4.46
100%	4.68	3.12	3.07	3.62

Table 4: Influence on the mean absolute error of amount of training data employed as percentage of total number of training examples in the Biwi dataset.

6. Conclusions

In this paper, we propose the coupling of a Gaussian mixture of linear inverse regressions with a ConvNet, we describe the methodological foundations and the associated algorithm to jointly train the deep network and the regression function, and we evaluate our model on the problem of head-pose estimation. From an experimental point of view, our contribution can be summarized as follows. First, we show that the proposed inverse regression model outperforms L_2-based regression models used by most of the state-of-the-art computer vision methods, at least in the case of head-pose estimation. Second, our method works effectively on relatively small training datasets, without the need of incorporating additional data, as it is often proposed in the literature. Lastly, our proposal outperforms state-of-the-art methods in head-pose estimation testing on the most widely used head-pose dataset. To the best of our knowledge, we are the first to propose an inverse regression approach to train a deep network. As future work, we plan to test our method on other computer vision problems, like facial keypoint detection or full body pose estimation, and extend the type of distributions used in our mixtures, as for example t-distributions to make the model more robust to outliers.

Acknowledgments

Funding from the European Union FP7 ERC Advanced Grant VHIA (#340113) is greatly acknowledged.

References

[1] B. Ahn, J. Park, and I. S. Kweon. Real-time head orientation from a monocular camera using deep neural network. In *ACCV*, pages 82–96, 2014. 2, 8

[2] V. Belagiannis, C. Rupprecht, G. Carneiro, and N. Navab. Robust optimization for deep regression. In *ICCV*, pages 2830–2838, 2015. 2

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.*, 5(2):157–166, 1994. 4

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages 886–893, 2005. 5

[5] A. Deleforge, F. Forbes, S. Ba, and R. Horaud. Hyper-Spectral Image Analysis with Partially-Latent Regression and Spatial Markov Dependencies. *IEEE Journal on Selected Topics in Signal Processing*, 9(6):1037–1048, Sept. 2015. 1

[6] A. Deleforge, F. Forbes, and R. Horaud. High-Dimensional Regression with Gaussian Mixtures and Partially-Latent Response Variables. *Statistics and Computing*, 25(5):893–911, 2015. 1, 3, 4, 6

[7] A. Deleforge, R. Horaud, Y. Y. Schechner, and L. Girin. Co-localization of audio sources in images using binaural features and locally-linear regression. *IEEE Transactions on Audio, Speech and Language Processing*, 23(4):718–731, 2015. 1

[8] V. Drouard, S. Ba, G. Evangelidis, A. Deleforge, and R. Horaud. Head pose estimation via probabilistic high-dimensional regression. In *IEEE ICIP*, pages 4624–4628, 2015. 1, 7

[9] V. Drouard, R. Horaud, A. Deleforge, S. Ba, and G. Evangelidis. Robust head-pose estimation based on partially-latent mixture of linear regressions. *arXiv preprint arXiv:1603.09732*, 2016. 6, 7, 8

[10] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Gool. Random Forests for Real Time 3D Face Analysis. *Int. J. Comput. Vision*, 101(3):437–458, 2013. 5, 7, 8

[11] J. Gan, L. Li, Y. Zhai, and Y. Liu. Deep self-taught learning for facial beauty prediction. *Neurocomputing*, 144:295 – 303, 2014. 2

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *CVPR*, pages 580–587, 2014. 1

[13] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016. 4

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 4

[15] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Reading text in the wild with convolutional neural networks. *Int. J. Comput. Vision*, 116(1):1–20, 2016. 7

[16] H. J. Kim, B. M. Smith, N. Adluru, C. R. Dyer, S. C. Johnson, and V. Singh. Abundant Inverse Regression Using Sufficient Reduction and Its Applications. In *ECCV*, pages 570–584, 2016. 1

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, pages 1097–1105. 2012. 1

[18] K.-C. Li. Sliced inverse regression for dimension reduction. *Journal of the American Statistical Association*, 86(414):316–327, 1991. 1, 3

[19] X. Li, L. Zhao, L. Wei, M.-H. Yang, F. Wu, Y. Zhuang, H. Ling, and J. Wang. Deepsaliency: Multi-task deep neural network model for salient object detection. *IEEE Trans. Image Process.*, 25(8):3919–3930, 2016. 1

[20] X. Liu, W. Liang, Y. Wang, S. Li, and M. Pei. 3D head pose estimation with convolutional neural network trained on synthetic images. In *ICIP*, pages 1289–1293, 2016. 2, 7, 8

[21] S. Miao, Z. J. Wang, and R. Liao. A CNN Regression Approach for Real-Time 2D/3D Registration. *IEEE Trans. Med. Imag.*, 35(5):1352–1363, 2016. 1

[22] S. Mukherjee and N. Robertson. Deep Head Pose: Gaze-Direction Estimation in Multimodal Video. *IEEE Trans. Multimedia*, 17(11):2094–2107, 2015. 2, 7, 8

[23] E. Murphy-Chutorian and M. M. Trivedi. Head pose estimation in computer vision: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(4):607–626, 2009. 2

[24] M. Osadchy, Y. L. Cun, and M. L. Miller. Synergistic face detection and pose estimation with energy-based models. *J. Mach. Learn. Res.*, 8:1197–1215, 2007. 2

[25] E. Perthame, F. Forbes, and A. Deleforge. Inverse regression approach to robust non-linear high-to-low dimensional mapping. Technical report, INRIA, 2016. 1, 3

[26] R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *CoRR*, abs/1603.01249, 2016. 2

[27] G. Riegler, D. Ferstl, M. Ruther, and H. Bischof. Hough Networks for Head Pose Estimation and Facial Feature Localization. In *BMVC*, 2014. 2

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vision*, 115(3):211–252, 2015. 1

[29] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *ICLR*. 2014. 1

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1, 6

[31] A. J. Smola and B. Schölkopf. A tutorial on support vector regression. *Stat Comput*, 2004. 5

[32] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In *CVPR*, pages 3476–3483, 2013. 2, 7

[33] 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*, pages 1–9, 2015. 1, 2

[34] A. Toshev and C. Szegedy. DeepPose: Human Pose Estimation via Deep Neural Networks. In *CVPR*, pages 1653–1660, 2014. 2, 7

[35] B. Wang, W. Liang, Y. Wang, and Y. Liang. Head pose estimation with combined 2D SIFT and 3D HOG features. In *ICIG*, pages 650–655, 2013. 7, 8

[36] H. Yang, W. Mou, Y. Zhang, I. Patras, H. Gunes, and P. Robinson. Face alignment assisted by head pose estimation. *CoRR*, abs/1507.03148, 2015. 2

[37] X. Zhen, Z. Wang, A. Islam, M. Bhaduri, I. Chan, and S. Li. Multi-scale deep networks and regression forests for direct bi-ventricular volume estimation. *Med Image Anal*, 30:120 – 129, 2016. 1, 2