Unsupervised Pose Estimation Across Domains

Arjun Karpur Qixing Huang

Summer 2017

Abstract

We attempt to solve the problem of pose/viewpoint estimation on 2D images without the presence of a large, well-labeled 2D or 3D dataset within our target domain. In order to accomplish this, we leverage our few available objects models to create 2D object renderings at known poses as a source domain, and learn pose estimation in our target domain images using the source domain images. We do so by attempting to learn a joint embedding to represent images from both domains/image spaces. We will rely on unsupervised clustering methods to learn this embedding. In this document, we detail the problem, introduce our initial methodology, and show results. Additionally, we analyze the results and provide future directions to improve accuracy and robustness.

1 Introduction

Within most data domains, one major issue with solving problems via machine learning techniques is the lack of openly-available and well-labeled big data. Traditionally, pose estimation of 2D images can be solved using annotated datasets, such as Pascal3D+ [6], and a combination of common machine learning and deep learning techniques to leverage the training labels and images to learn a pose-sensitive image embedding. When working in a domain without the presence of well-labeled data (which is a common problem in the medical domain), we must instead rely on alternative methods and datasets to accomplish our goals.

Our team is working on human skull CT scan/3D model data, motivated by ongoing work by doctors at the UT Dell Medical School. Unfortunately, the partnering hospital has only been able to provide our team with roughly 60 patient models due to various logistical, privacy, and collection issues, and thus we must perform any learning using this limited dataset. We have additionally scraped roughly 10,000 clean 2D ('real') images of human skulls from the internet using Bing Image Search API, and our goal is to perform pose estimation on these 10,000 real images. We plan to leverage 2D renderings of our 3D skull models ('rendered' images), but must do so in a way that accounts for difference in image domains between the real and rendered images (see Figure 1). Training using data from a single domain may not yield accurate results when testing in alternate domains.

Others, such as [5], have approached this problem by introducing noise to the 2D renderings of the object models, although we argue that this technique is not suitable in image domains where it may be hard to manually model the 'real' image noise variability and distribution. Additionally, [5] relies heavily on the availability of large numbers of 3D models for the specific domain (cars, chairs, etc...), which we do not have at our disposal.

We propose learning a joint embedding function between the real images and rendered views of the 3D models that is domain-invariant yet pose sensitive, such that images of similar poses will have similar embeddings regardless of if they are real or rendered images. We will carry out this task through unsupervised methods, specifically clustering on a pre-trained embedding and attempting to fine-tune this pre-trained network. Our hope is that by learning a domain-invariant embedding, we can generate a large skull dataset that is labeled across multiple categories using images from multiple different image domains (rendered, real, drawings, etc...). This will ideally allow for and encourage the utilization of machine learning and deep learning techniques in fields that currently do not have large datasets at their disposal, as in our skull medical domain. This additionally can reduce the time required to gather large datasets in a specific domain before learning can begin.
Figure 1: Examples of skull images used for this project. The first two images are examples of 'real' skull images scraped from the Internet. The third image is an example of a skull model 'rendered' image that was rendered at a known pose.

Note: As indicated by the results in Section 4, our technique did not see major improvements in pose estimation accuracy for the given set of 2D images. Our results, however, did provide useful insights into the problem of pose estimation and transfer learning, and has provided us with inspiration for future work. The remainder of the report is organized as follows: Section 2 defines our pose estimation task formally, Section 3 presents our unsupervised method of solving pose estimation, Section 4 presents empirical evaluations of our methodology, Section 5 gives analysis of our results, and Section 6 provides areas of future work.

2 Problem Statement

Let us define a pose estimation function space $\mathcal{F}$ that provides a pose estimate in terms of azimuth angle $\theta$ and elevation angle $\phi$, given an image $x$ of width $i$, height $j$, and depth $k$ (for RGB, $k = 3$). Thus, let $\mathcal{F} : \mathbb{R}^{i \times j \times k} \rightarrow \mathbb{R}^2$ s.t. for any $f \in \mathcal{F}$, $f(x) = p_x = (\theta_x^e, \phi_x^e)$ where $p_x^e$ is some estimate of the true pose of $x$. Suppose for image $x$ the true pose is $p_t^x = (\theta_t^x, \phi_t^x)$. Thus, for a set of $N$ real images, we would like to find...

$$\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \text{dist}(f(x_i), p_i^t) = \sum_{i=1}^{N} \text{dist}(p_i^e, p_i^t)$$

where $\text{dist}(\cdot, \cdot)$ is some distance metric between two poses (ex: L2 distance, great-circle distance, etc.). In other words, we would like to find an embedding function within the pose estimation function space that provides estimates as close to the ground truth as possible for our set of real images.

3 Method

Our technique uses fine-tuning of a pre-trained network to refine an embedding function for a given skull image. We first render a number of images from our set of 3D models at known poses, and use these images to compare against the real images for a rough, initial pose estimation. We then use these pose estimations and image distances to generate a large number of training examples in the form of 'triplets'. We perform training by fine-tuning our network using these triplets and a triplet loss function that utilizes both real and rendered images. By training using real and rendered images, we hope to learn an embedding that accurately represents both image domains. Lastly, we use our new embedding function (after fine-tuning) to perform pose estimation again and check for improvements.

3.1 Pose Estimation via Nearest Neighbor

Given a real image $x$, we can perform pose estimation using an embedding function $F(x)$ and a set of rendered images $R$. We first render the $M$ skull models in $N$ known poses, which gives us the set $R$ ($r_{ij} \in R$.}
Figure 2: AlexNet architecture as shown in [3]. We mimic the same architecture and simply add an L2
Normalization layer after the last fully connected layer (not pictured).

∀i ∈ N, j ∈ M). For a given pose i, we can compute the distance between the real image’s embedding
(or feature vector) F(x) and the embedding of each model in pose i, {F(r_i1), F(r_i2), ..., F(r_iM)}. We can
construct a list of distances D_i for pose i between the real and rendered images such that...

\[ D_i = \{ ||F(x) - F(r_{i1})||_2^2, ..., ||F(x) - F(r_{iM})||_2^2 \} \]

(2)

We can then estimate the pose for real image x as...

\[ p_x^e = \min_i \sum_{j=1}^{5} \text{sort}(D_i)_j \]

(3)

It is essential to have an accurate embedding function that is invariant to the image domain, meaning
that it results in similar embeddings regardless for real/rendered images in similar poses. Additionally, we
only consider the top 5 in order to ignore outliers and noise in the data and focus only on a few closest
rendered views.

3.2 Triplet Generation

As motivated by [4], we use data triplets (consisting of real and rendered images) as training data to fine-
tune our network. Given our network and some learned network weights, we can generate an embedding for
each of the real and rendered images that are then used for similarity comparisons during triplet generation.
For these experiments, we generate all triplets offline before training time using network weights pre-trained
on ImageNet (see 'AlexNet-fc8' in Section 3).

We define a triplet as a real image (anchor), a similar rendered image (positive), and a dissimilar rendered
image (negative). To generate a triplet, we first select a real image to be used as an anchor. We then perform
pose estimation on the image and select one of the 5 closest renderings of models in the predicted pose to use
as a positive image example. We then sample a random rendered image from any pose and model (except
for the top 5 positive examples in the predicted pose) to use as a negative example. For each anchor and
positive pair, we can select multiple negative examples to provide our network with a robust set of training
examples.

3.3 Learning an Embedding

Training a network from scratch often requires massive amounts of unique training examples to robustly
learn the task at hand. Due to the limited data available to us (~90 skull models and ~10,000 real images),
we must rely on fine-tuning to learn our joint embedding. Fine-tuning allows us to leverage information learned from other domains as a starting point when learning an embedding that completely represents a skull image. Our network is based closely on the AlexNet architecture [3] as shown in Figure 2, with only a single layer appended for L2 normalization (explained in Section 3.3.1). At training time, we can initialize our network using weights that have been determined through training on ImageNet so that our network can refine previously learned information rather than learn from scratch. We use a 'Triplet Loss Function' to encourage similarity between the anchor and positive while encouraging dissimilarity between the anchor and negative.

We use the Caffe Deep Learning Framework to implement the convolutional neural network and perform fine-tuning over our training set [2].

3.3.1 Triplet Loss Function

The loss function that we use attempts to maximize the distance between the anchor and the negative example while minimizing the distance between the anchor and the positive example [4]. Given a single triplet \((a_i, p_i, n_i)\), an embedding function \(F(x)\), and some margin \(\alpha\), we can define the loss as

\[
L(a_i, p_i, n_i) = \max(\|F(a_i) - F(p_i)\|^2_2 - \|F(a_i) - F(n_i)\|^2_2 + \alpha, 0)
\]

We determine the pose through a nearest neighbor approach, so enforcing some sort of margin \(\alpha\) encourages separation between similar and dissimilar pose clusters. Because of the margin, L2-normalization is required so that absolute distances (in relation to the margin) are relevant. By constraining the norm of an embedding to 1, we place all embeddings onto an N-dimensional hypersphere in which a concrete margin of 0.2, for example, has a significant influence on the training of the embedding.

4 Results

After learning a new embedding function for skull images, we can perform pose estimation on a subset of test real images to test the effectiveness of our methodology. Additionally, we test the accuracy of predicting generated triplets to verify that our training methodology is valid. For testing the accuracy of triplet prediction and pose estimation, we compare three different learned embeddings, which are:

- **AlexNet-fc8** - 1000 dimension embedding from the final fully connected layer ('fc8') of the network pretrained on ImageNet (360,000 iterations). No fine-tuning on the skull data is used for this embedding. Pre-trained network weights from: [https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet](https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet)
- **1.28mil** - 1000 dimension embedding from the network with an added L2 Normalization layer after the final fully connected layer. After initializing the network with weights trained on ImageNet (same as 'AlexNet-fc8'), we performed fine-tuning using triplet loss on roughly 1.28 million triplets (10300 anchors, 5 positives per anchor, 25 negatives per positive). Training took approximately 4 hours with all triplets generated offline before training time using the AlexNet-fc8 embedding.
- **23.2mil** - Same as '1.28mil', but trained on roughly 23.2 million triplets (10300 anchors, 5 positives per anchor, 450 negatives per positive). Training took approximately 72 hours.

Code for the experiments can be found at the following link: [https://github.com/arjunkarpur/unsupervised-2d-pose-estimation](https://github.com/arjunkarpur/unsupervised-2d-pose-estimation)

4.1 Triplet Prediction

We perform testing on triplet prediction to verify that our fine-tuning technique is learning network weights that lead to accurate embeddings. We generate a set of roughly 200,000 unique triplets to be used for testing purposes. In Table 1, we present the error rate, average loss per error, and average loss across all training triplets for each of the three embeddings. Error rate represents the percentage of test triplets
Table 1: Testing accuracy of embedding on triplet loss function (trained with margin $\alpha = 0.2$)

<table>
<thead>
<tr>
<th>Embedding Name</th>
<th>Error Rate</th>
<th>Avg Loss (only errors)</th>
<th>Avg. Loss (all triplets)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet-fc8</td>
<td>30.6%</td>
<td>0.119</td>
<td>0.0360</td>
</tr>
<tr>
<td>1.28mil</td>
<td>0.47%</td>
<td>0.159</td>
<td>0.0007</td>
</tr>
<tr>
<td>23.2mil</td>
<td>0.27%</td>
<td>0.185</td>
<td>0.0005</td>
</tr>
</tbody>
</table>

Table 2: Pose estimation accuracy using the different learned embeddings. (trained with margin $\alpha = 0.2$)

<table>
<thead>
<tr>
<th>Embedding Name</th>
<th>Avg. Azimuth Error</th>
<th>Avg. Elevation Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet-fc8</td>
<td>43.9°</td>
<td>16.9°</td>
</tr>
<tr>
<td>1.28mil</td>
<td>41.7°</td>
<td>13.1°</td>
</tr>
<tr>
<td>23.2mil</td>
<td>45.5°</td>
<td>13.1°</td>
</tr>
</tbody>
</table>

4.2 Pose Estimation

Out of the 10,300 real images scraped from the Internet, we set aside 109 and manually labeled their poses to serve as a testing set. We perform pose estimation on the real images in the test set using the three different embeddings to test the improvement in the pose estimation task due to fine-tuning over the generated triplets. The accuracy is measured in terms of average azimuth and elevation errors in degrees. Pose estimation for AlexNet-fc8 uses the un-normalized embedding, as this is what the ImageNet pre-training learns directly. These results are presented in Table 2.

4.3 Margin Experiments

This section shows the results of using different margins for the triplet loss function. Because our embeddings are normalized using L2 Normalization, all points (embeddings) in this space must lie on the N-dimensional hypersphere. On this hypersphere, we would like similar poses to cluster together closely and for these clusters to be evenly distributed across the sphere to allow for more accurate nearest neighbor estimates. The maximum distance between any two points on the hypersphere is 2, and thus enforcing a large margin $\alpha$ leads to sufficient space between pose clusters.

The effects of modifying $\alpha$ can be seen in Table 2. We measure the effects of modifying margin on pose estimation, as well as the average distances between training examples and different clusters. 'Avg. Pose Distance' refers to the average distance between a test image and its estimated pose cluster, while 'Avg. Non-Pose Distance' refers to the average distance between a test image and all other pose clusters. These values are averaged across the entire testing set.

5 Analysis

As evident from our results shown in Table 1, our training methodology successfully learns the pattern defined by our pre-generated triplets. Unfortunately, this doesn’t appear to lead to more accurate pose estimation (Table 2). The triplets are generated using the un-normalized AlexNet ’fc8’ embedding while triplet accuracy is measured using the normalized embedding, which explains the significant triplet prediction error rate for 'AlexNet-fc8'. After training, the triplet error rate drops significantly while pose estimation accuracy stays roughly constant. Therefore, it is reasonable to interpret these results as the network simply learning an L2-normalization of the previous embedding.
<table>
<thead>
<tr>
<th>Margin (α)</th>
<th>Avg. Pose Distance</th>
<th>Avg. Non-Pose Distance</th>
<th>Azimuth Error</th>
<th>Elevation Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>0.69</td>
<td>1.31</td>
<td>41.7°</td>
<td>13.1°</td>
</tr>
<tr>
<td>0.4</td>
<td>0.65</td>
<td>1.49</td>
<td>43.0°</td>
<td>12.0°</td>
</tr>
<tr>
<td>0.6</td>
<td>0.61</td>
<td>1.58</td>
<td>43.0°</td>
<td>11.7°</td>
</tr>
<tr>
<td>0.8</td>
<td>0.51</td>
<td>1.62</td>
<td>42.9°</td>
<td>10.9°</td>
</tr>
<tr>
<td>1.0</td>
<td>0.45</td>
<td>1.66</td>
<td>43.7°</td>
<td>10.9°</td>
</tr>
<tr>
<td>1.2</td>
<td>0.39</td>
<td>1.67</td>
<td>47.0°</td>
<td>11.2°</td>
</tr>
<tr>
<td>1.4</td>
<td>0.33</td>
<td>1.68</td>
<td>46.6°</td>
<td>10.6°</td>
</tr>
</tbody>
</table>

Table 3: The effects of training with varying margin values α. The network was trained using the 1.28mil training triplets with the same learning parameters.

These results show the networks ability to learn the provided triplet data properly, so it is likely that the failure in methodology comes from our triplet generation technique. Millions of triplets are generated using the initial embedding function, so it makes sense that our network simply mimics this function. The difference between the source and target image domains is likely a large barrier that prevents accurate initial pose estimation, leading to poor triplets and poor accuracy.

6 Next Steps

There are a few next steps we can take in order to explore this field more and try to achieve better pose estimation results.

- **Use alternate triplet generation techniques**: We tried using multiple different triplet generation techniques, including the one presented in this document, without seeing much improvement in accuracy. One future direction would be to generate triplets more intelligently and robustly, such as adding variability and noise to positive and negative selection to account for the inaccuracy of the embedding function prior to fine-tuning.

- **Perform online triplet generation**: If we were to perform triplet generation online, we may see the network learning positive and negatives examples itself and experience better pose estimation results. The authors of [7] were able to improve accuracy in their task by alternating between clustering and training, so a similar technique in our domain may also see success.

- **Experiment with hyper-paramters**: There are a number of parameters that can be fine-tuned in this method, such as the triplet loss margin, 'Top N' metric for pose estimation (N=5 currently), and various training parameters (learning rate, weight decay, etc...). Performing cross-validation tests would allow us to find the optimal set of hyper-parameters, although it is doubtful that this will provide significant benefits to our method.

- **Use Transfer Learning/Domain Adaptation**: The most probably source of failure is the difference in image domains between the real and rendered images. Crossing this domain gap is essential to using source images as training data for target images. The fields of Transfer Learning (and particularly Domain Adaptation) focus entirely this issue and use a multitude of either shallow or deep techniques to accomplish this transfer [1]. In our opinion, this is the most promising direction, and we are now focusing our efforts on using Domain Adaptation to accomplish pose estimation.

References


