Real-Time Head Pose Estimation with Convolutional Neural Networks

Zhiang Hu
zhianghu@stanford.edu

Abstract

In this project, we seek to develop an accurate and efficient methodology to address the challenge of real-time head pose estimation. Though there have been previous attempts to apply convolutional neural networks to this fundamentally subtle task, to the best of our knowledge, there exist no initiatives that have directly utilized Convolutional Neural Networks (CNNs) in tandem with these methods for a more comprehensive, thorough approach.

This project thus proposes a methodology that employs CNNs to predict the location and orientation of a head in 3D space. We pursue classification and regression at various degrees of fineness on high-quality data captured from depth sensors, and evaluate our models by comparing our performance to that obtained by state-of-the-art systems.

1. Introduction

The idea of head pose estimation is an inherently rich yet open-ended task. In contrast to the well-explored domains of face detection and recognition [8, 9], head pose estimation additionally poses a unique challenge to computer vision systems in that it is identity invariant, and there are fewer thoroughly evaluated systems that have been tested within large ranges of contexts [2].

It also represents an integral part of a more refined framework for 3D understanding, and contains a diverse range of applications that are important in the analysis of human behavior. For instance, one immediate application of head pose estimation is the establishment of a driving assistance system, where changes in the driver's head pose or gaze may yield information about the driver's state of awareness. This information in turn could be used to give feedback to the driver about the presence of unsafe behavior.

There have been several approaches in the past that have applied machine learning algorithms to the problem of head pose estimation. Approaches based on 2D images have focused on either analyzing the entire facial region, or relied on the localization of specific facial features (such as nose tip). In 2D appearance-based methods, the head pose space is often discretized, and separate detectors are learned for subsets of poses [9]. Approaches that use 2D images are generally highly sensitive to variation in illumination, shadows, and occlusions, and are suitable for fixed, predictable surroundings. Recently, the advent of 3D acquisition systems has allowed these problems to be overcome with the use of additional depth information. Despite these advancements, however, most existing 3D methods are neither robust to wide variations in head pose nor in real time.

Breitenstein *et al.* [7] developed an algorithm for automatic and real-time face pose estimation from 3D. Their method involves offline pre-computation of reference pose range images and the acquisition of range images of faces. Head pose hypotheses are obtained from these images by computing candidate nose positions and orientations. However, this method requires signifcant use of a GPU to compare the reference pose range images to the input range image and thus cannot generalize to applications where hardware constraints limit computational resources.

Fanelli *et al.*, in [4], presented one of the first approaches to obtain actual real-time head pose estimation using random regression forests from depth data. This initiative does not rely on specific hardware such as a GPU, and obtained superior performance on a publicly available dataset.

There have also been previous introductory methods that have utilized Convolutional Neural Networks in order to simultaneously estimate face detection and orientation in real time. However, these were applied only to a specific range of testing conditions [6].

However, to the best of our knowledge, there have been no approaches that have made significant use of Convolutional Neural Networks (CNNs) for a more comprehensive analysis of head pose estimation. We thus present a methodology utilizing CNNs for obtaining an efficient, accurate estimate of head pose from a single image in real time, under a broad range of testing conditions.

2. Technical Approach

In this section, we begin our discussion of the various algorithms we apply to the challenge of real-time head pose estimation. Our approach to this task is twofold: we utilize Convolutional Neural Networks (CNNs) with depth data captured by a sensor to compute the location and orienta-

tion of a head in 3D space.

2.1. Dataset

In this project, we propose to use two dataset: a) the depth dataset from Fanelli which is captured by kinect and b) synthetic dataset which is generated using OSGRenderer and CAD model from 3dwarehouse.

The kinect dataset is presented by Fanelli *et al.*, in [4]. This dataset contains around 15,000 depth images with their head centers and head directions.

Another dataset is generated by CAD model from 3Dwarehouse. We have created 920,000 images with different head centers and head poses. Due to the limitation of time, we are only using the kinect dataset to train the CNNs. The Synthetic dataset will be used in the future.

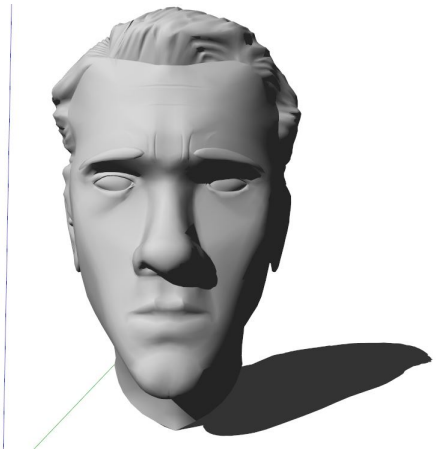

Figure 1: CAD model of head

2.2. Convolutional Neural Network

Convolutional Neural Networks have recently become an increasingly popular tool in computer vision related applications, often achieving state-of-the-art performance in classification, regression, and segmentation. There has in fact even been work applying CNNs to broader forms of human pose estimation [10] while achieving superlative performance. Thus, we believe that CNNs will be inherently suited to our task of head pose estimation.

We now describe the way we configure the structure of our network. Our input data once again consists of depth images captured from a 3D sensor. Our network architecture will be based on existing networks that are pre-trained on the ImageNet database by Krizhevsky *et al.* [1]. We can use these existing implementations to both improve efficiency and accuracy in our predictions, since we can avoid having to retrain the network on large amounts of new data while making use of the weights learned from ImageNet. We now present several ways of leveraging Convolutional Neural Networks towards head pose estimation.

Method 1: The first approach would be to define the ImageNet CNN as a *regressor* built on the same structure as ImageNet network. We can accomplish this by simply modifying the last fully connected layer (fc8-conv) of the pre-existing network into a fully connected layer that reports only 6 numbers (*x, y, z, yaw, pitch, roll*) instead of the original 1000, corresponding to the number of classes in ImageNet. We can also simultaneously change the desired loss function from the standard Softmax-based loss to Euclidean loss, and retrain the new last layer on our own dataset (which we later describe in our results). The structure of ImageNet CNN is shown below, with euclidean loss at the end of network.

$$(Conv-Relu-Pool-LRN)\times 2-(Conv-Relu)\times 3-Pool$$
$$-(FC-Relu-Dropout)\times 2-FC$$

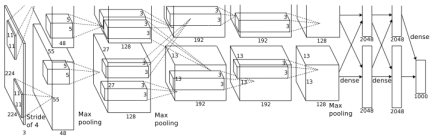

Figure 2: ImageNet Structure

Method 2: We could maintain the existing network structure, and instead of posing a regression problem, we could formulate a *classification* challenge by discretizing possible head poses/locations into a grid of small bins, using the ImageNet CNN structure. We could then keep the same loss function but replace the last layer with a fully-connected layer with an output dimension equal to the number of bins. Thus, we once again reuse the structure of ImageNet to predict categories of head pose.

Method 3: A third approach would be to use *Shallowed ImageNet CNN structure* with fewer hidden layers to produce a prediction. The motivation of using this shallowed version of ImageNet CNN is that: the kinect dataset has only about 15,000 images, therefore a too deep network would very easily overfit; 2) the task is to predict exact information out of an image, while the forward pass of pooling layers is an information loss process, with less pooling layers, the variance of prediction would be smaller. The structures of the shallowd CNNs can be from two conv-relu-pool sandwiches to four conv-relu-pool sandwiches. Below is an example of two-sandwiched shallowd ImageNet CNN.

$$(Conv-Relu-Pool-LRN)\times 2-(FC-Relu-Dropout)\times 2-FC$$

3. Experiments

In this section we will describe the experiments we have done for the three methods.

3.1. ImageNet as Regressor

We now discuss the results we have obtained from performing initial testing with Convolutional Neural Networks.We have completed the first step of **Method 1** and , in which we use a single ImageNet CNN fine tuned from pre-trained Imagenet model as a regressor to predict the complete pose of the head. Figure below shows the visualization of predictions.

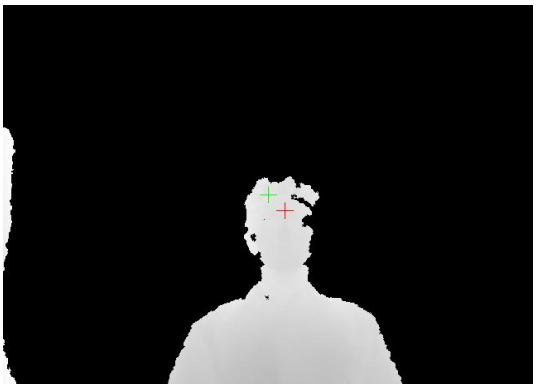

Figure 3: Example of prediction of a testing image

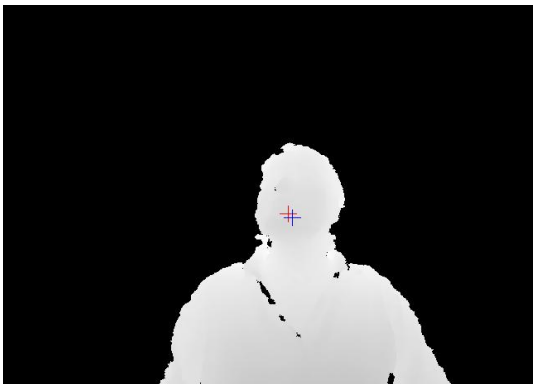

Figure 4: Example of prediction of a training image

However, for this report, we have first run preliminary testing on locating the center of the head alone (we will soon extend this into the actual orientation estimation, but for now we only output the 3 coordinates). For our input data, we train the new last layer of the network on a large portion of the same non-synthetic dataset provided by Fannelli *et al.* [4], as described earlier. We then test on a separate portion of the data. The accuracy of our results with respect to different thresholds is shown in the figures below.

As we can see, there is a huge gap between training and testing accuracy, which implies overfitting. This might be due to the lack of sufficient data, or the need for a more

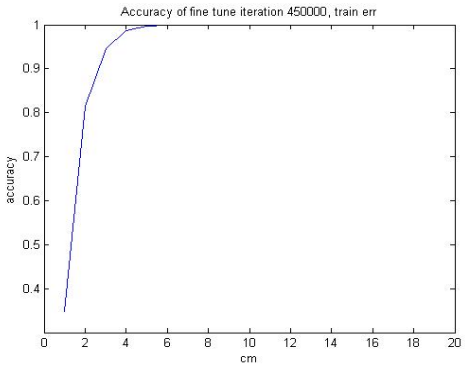

Figure 5: Training accuracy vs. threshold (cm).

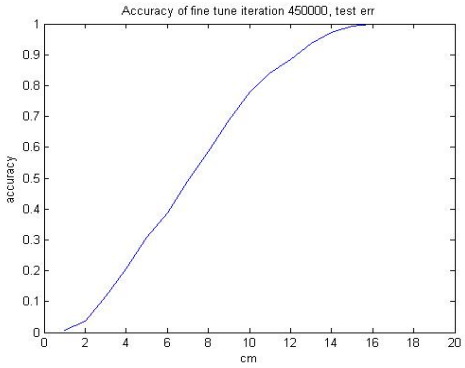

Figure 6: Testing accuracy vs. threshold (cm).

select choice of hyperparameters and layer modifications. There are several techniques that are currently being applied to compensate for this, including increasing dropout rate and enhancing the amount of regularization. We also are attempting method 2 to see if discretization of our output space will yield better results. In any case, it is clear that we must first address the challenge of head center localization before moving on to head orientation estimation. Figure 7 to 16 show the distribution of the error vector's length for the ImageNet CNN structure fine tuned with different regularization terms.

As we can see, when regularization (Weight Decay in Caffe) is about 1000, both testing and training distibution have peak at around 7 cms.

3.2. ImageNet as Classifier

We have tested the classifier version of ImageNet network on two kind of discretizations: 1) splitting the head

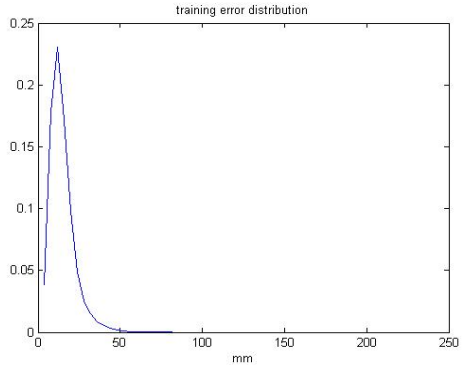

Figure 7: training distribution of error vector lengths for regularization = 1

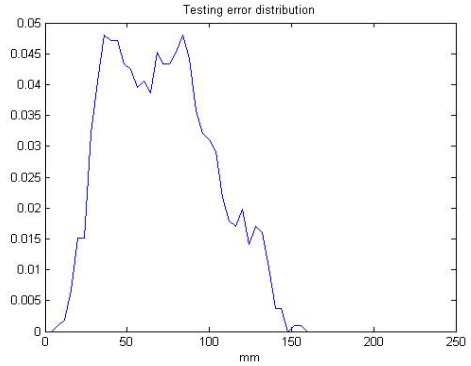

Figure 8: testing distribution of error vector lengths for regularization = 1

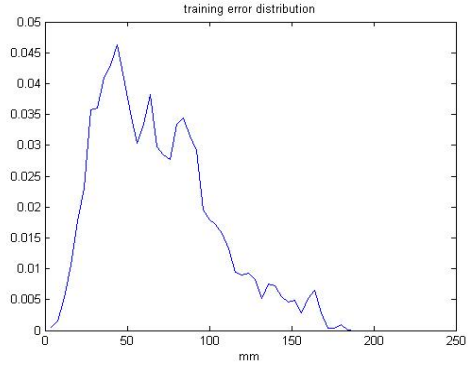

Figure 9: training distribution of error vector lengths for regularization = 500

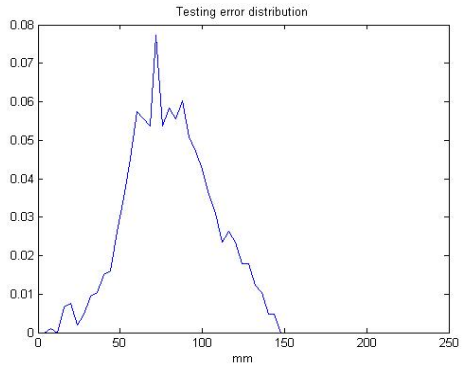

Figure 10: testing distribution of error vector lengths for regularization = 500

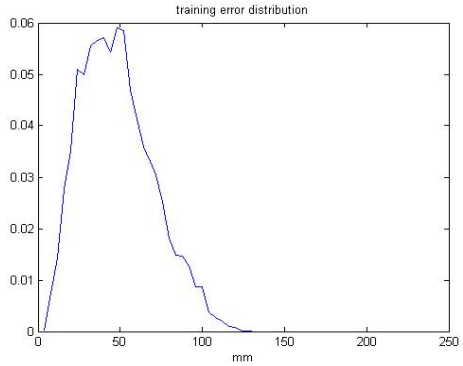

Figure 11: training distribution of error vector lengths for regularization = 1000

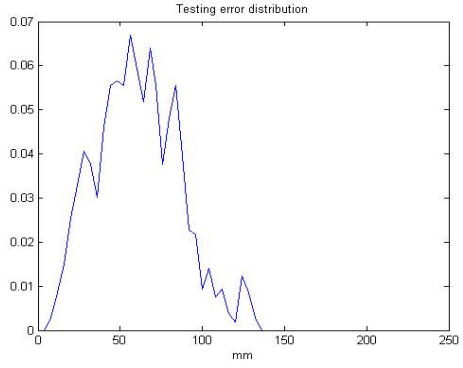

Figure 12: testing distribution of error vector lengths for regularization = 1000

center space into 32 cubes and 2) splitting the head center space into two halfspaces.

The structure of the network is shown below.

$$(Conv-Relu-Pool-LRN)\times 2-(Conv-Relu)\times 3-Pool$$

$$-(FC-Relu-Dropout)\times 2$$

$$-FC$$

For the first kind of 32 categories, figures 17 and 18 below shows the visualization of confusion matrices. As shown in the plots, both in training and testing there is bias

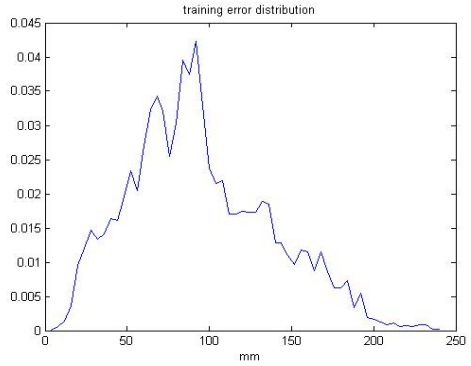

Figure 13: training distribution of error vector lengths for regularization = 3000

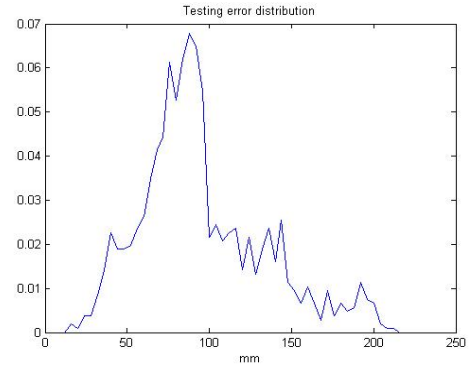

Figure 14: testing distribution of error vector lengths for regularization = 3000

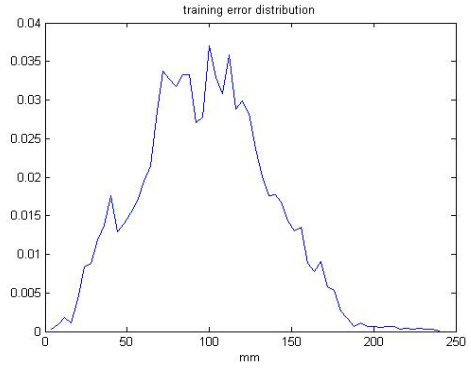

Figure 15: training distribution of error vector lengths for regularization = 10000

towards some categories.

For the second discretization, we splitted the whole space into up and down with respect to the vertical axis. The experimental results shows that the training accuracy is 73%, but the testing accuracy only has 47%. Also, the

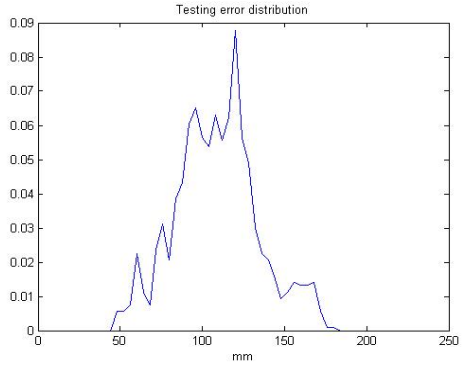

Figure 16: testing distribution of error vector lengths for regularization = 10000

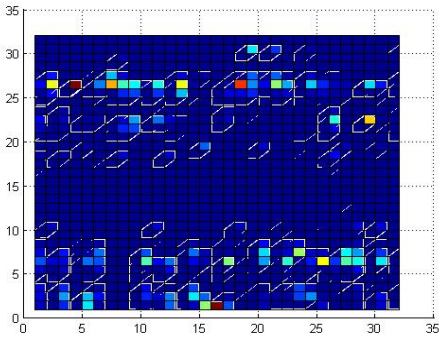

Figure 17: visualization of confusion matrix for training classifer, 32 classes

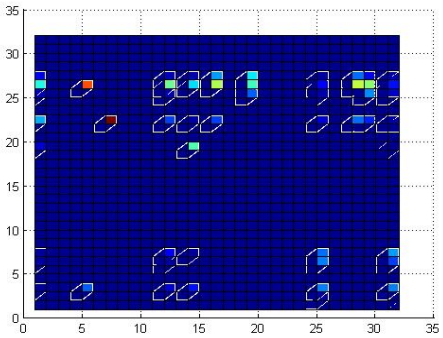

Figure 18: visualization of confusion matrix for testing classifier, 32 classes

confusion matrix for testing shows that the CNN is predicting all testing dataset as category up. This result implies that, with the limited information provided to the CNN, the model suffers serious overfitting problem.

3.3. Shallowed ImageNet

Currently we have done experiment for two conv-relu-pool sandwiched version of shallowed ImageNet network. The structure of the tested network is shown below.

$$(Conv-Relu-Pool-LRN)\times 2-(FC-Relu-Dropout)\times 2 \\ -FC$$

The network is fine tuned from ImageNet network, reusing the first two convolutional sandwiches. The input dataset of this shallowd network is the kinect dataset as mentioned earlier, with 14,000 images as training set, 1,000 images as testing set.

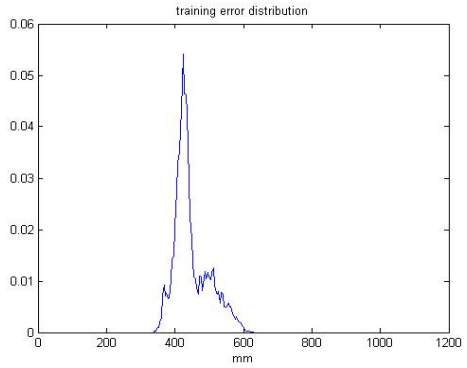

Figure 19: training distribution of error vector lengths for 2-sandwiched ImageNet Network

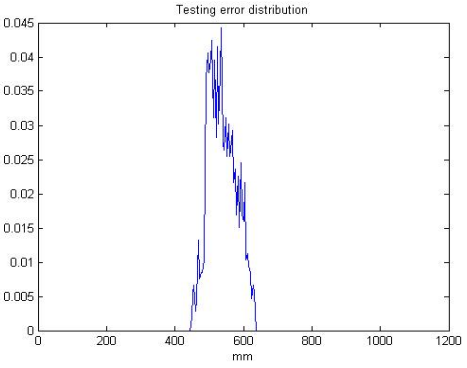

Figure 20: testing distribution of error vector lengths for 2-sandwiched ImageNet Network

This result shows that two sandwiches of conv-relu-pool is too shallow for the dataset, which leads to heavy underfit. To address this, we will reduce the dropout and the fully connected layers, as well as increase the depth of the network.

4. Conclusion

We have currently implemented and tested several structures of CNNs to detect the head center. It is important to find the balancing point of the complexity of CNN so that the network can learn useful features out of the depth images to predict the head poses. A too deep CNN can overfit our tried dataset easily and lose position information because of the pooling layers, while a too shallow network will result in underfitting.

4.1. Future Work

The experiments show that the ImageNet CNN is too complicated for the kinect dataset and 2-sandwiched network is too simple, therefore more structures of shallower versions of CNN will be trained and tested, for example, 2 conv-relu-pool network without dropout layers, and three or four sandwiched networks.

Also, we have just successfully generated synthetic data from the CAD head model. The ImageNet CNN, as well as other structures, will be trained and tested on the new dataset. One possible approach would be to use *Cascade Convolutional Neural Networks* to produce a more accurate prediction. Inspired by the research completed by [10], this approach seeks to partition off the challenge of head pose estimation into separate networks, where each network might be based on a structure similar to ImageNet. The first network can produce an approximate guess as to values of head pose and location, and this can be used to perform a broad crop on the original image in order to focus purely on the head. This new image can be fed into the second network to obtain an even more precise prediction which can further inform the first network. We can thus continue the exchange between the two networks in this way until prediction converges.

Additionally, we are currently making use of pure depth data – it would be interesting to experiment with fusing depth and RGB images and making a prediction based on this joint information. Another approach we might later take is the use of multiple sensors from different viewpoints, in which we average out predictions from both the view points (after normalizing the estimations appropriately). Since the frames from our test sequence are continuous, we could also apply forms of temporal constraints by adding a tracking-based mechanism – however, our accuracies based on single images would need to be higher before attempting this. An ultimate compelling possibility would be to delegate head center detection to the random forest, and use this to inform the pose estimation output of the CNN.

In any case, however, our eventual goal is to finally train and produce an algorithm that excels in real time head pose estimation and is sensitive enough to be placed in a large range of real-world applications.

References

[1] A. Krizhevsky, I. Sutskever, and G.E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] E. Murphy-Chutorian and M.M. Trivedi. Head Pose Estimation in Computer Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 607–626.

[3] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Van Gool. Random Forests for Real Time 3D Face Analysis. *International Journal of Computer Vision*, 101(3):437–458.

[4] G. Fanelli, T. Weise, J. Gall, and L. Van Gool. Real Time Head Pose Estimation from Consumer Depth Cameras. *33rd Annual Symposium of the German Association for Pattern Recognition*, September.

[5] https://github.com/irvhsu/HeadPose.

[6] M. Osadchy, Y.L. Cun, and M.L. Miller. Synergistic Face Detection and Pose Estimation with Energy-Based Models. *Journal of Machine Learning Research*, 8, 2007.

[7] M.D. Breitenstein, D. Kuettel, T. Weise, and L. Van Gool. Real-Time Face Pose Estimation from Single Range Images. *Computer Vision and Pattern Recognition*, 2008.

[8] P. Sinha, B. Balas, Y. Ostrovsky, and R. Russell. Face Recognition by Humans: Nineteen Results All Computer Vision Researchers Should Know About. *Proceedings of the IEEE*, 94(11):1948–1962, 2006.

[9] P. Viola and M.J. Jones. Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2):137–154, 2004.

[10] A. Toshev and C. Szegedy. Deeppose: Human pose estimation via deep neural networks. *CoRR*, abs/1312.4659, 2013.

5. Supplementary materials

code for calculating distibution

```
step = 4;
Tot = 300;

fname_test_pref = '';
fname_train_pre = '';

fname_test = fname_test_pref;
data = hdf5read(fname_test, '/data');
label = hdf5read(fname_test, '/label');
num = size(data,4);
diff = zeros(3,num);
diff(:,:) = data(1,1,:,:) - label(1,1,:,:);
dist = sqrt(diag(diff'*diff));

acc = zeros(Tot,1);
for i = 1:Tot
    %acc(i) = sum((1-1)*10<dist & dist<i*10);
    idx = dist < i * step;
    idx = dist(idx) > (i-1)*step;
    acc(i) = sum(idx);
end
figure
x = 1*step:step:Tot*step;
plot(x, acc/num);
xlabel('mm'); % x-axis label
title(strcat('Testing error distribution'));

% training error

num_train = sum(hdf5_sizes(1:22)) ;
% offset: 1, 6, 11, 16
offset = [ 0, sum(hdf5_sizes(1:5)), sum(hdf5_sizes(
diff_patch = zeros(5000,4);
for idx_file = 1:3
    fname = strcat(fname_train_pre,int2str(idx_file
    data = hdf5read(fname, '/data');
    label = hdf5read(fname, '/label');
    num = size(data,4);
    diff = zeros(3,num);
    diff(:,:) = data(1,1,:,:) - label(1,1,:,:);
    diff_patch_tmp = sqrt(diag(diff'*diff));
    diff_patch(:,idx_file) = diff_patch_tmp(:,1);
end

dist_train = zeros(num_train,1);
curr_patch = 1;
j = 1;
for i = 1 : num_train
    j = i - offset(curr_patch);
    while (j>5000)
        curr_patch = curr_patch + 1;
        j = i - offset(curr_patch);
    end
    dist_train(i) = diff_patch(j,curr_patch);
end

acc = zeros(Tot,1);
for i = 1:Tot
    acc(i) = sum((i-1)*step<dist_train & dist_train
end
figure
plot(x,acc/num_train);
xlabel('mm'); % x-axis label
title(strcat('training error distribution'));
```

code for discretization

```
locations = [];
input_name_pref = 'C:\Users\harvyhu\Desktop\Ford\H
for iter_idx = 1:24
    fname = strcat(input_name_pref, 'depth_data_',i
    label = hdf5read(fname, '/label');
% size: 3 N
    locations = [locations, label];
```

```
end

K = 2;
[idx, C] = kmeans(locations', K);
colors = ['y+', 'k+', 'b+', 'b*', 'g+', 'b+', 'm*', 'c*', 'r*'];
figure
hold on
for i = 1:K
    tmp_idx = idx ==i;
    plot3(locations(1,tmp_idx), locations(2,tmp_idx), locations(3,tmp_idx),...
        colors(i));
end
hold off


Ass = zeros(size(locations,2),1);
for i = 1:size(locations,2)
    Ass(i) = compute_cat(locations(1,i), locations(2,i), locations(3,i), C, K);
end

% Convert head center


for iter_idx = 1:1
    fname = strcat(input_name_pref, 'depth_data_',int2str(iter_idx),'.h5');
    fname_out = strcat(output_name_pref, 'depth_data_',int2str(iter_idx),'.h5');
    data = hdf5read(fname, '/data');
    label = hdf5read(fname, '/label');
    % size of data should be 227 227 3 N
    new_label = zeros(1,size(label,2));
    for i = 1:size(label,2)
        new_label(1,i) = compute_cat(label(1,i), label(2,i), label(3,i), C, K);
    end
    hdf5write(fname_out,'/data',data,'/label', new_label);
end
```

code for computing scores

```
st = 1;
en = 1;

addpath('/opt/caffe-master/matlab/caffe');
    % read true train labels
    % read true val labels

    true_label_train = [];
    for folder_idx = 1:22
        fname =
         strcat('/scail/data/group/cvgl/u/zhianghu/headpose_data_227/depth_data_',int2str(fold
        curr_true_label = h5read(fname, '/label');
        true_label_train = [true_label_train, curr_true_label];
    end
    true_label_val = []
    for folder_idx = 23:24
        fname = strcat('/scail/data/group/cvgl/u/zh
        curr_true_label = h5read(fname, '/label');
        true_label_val = [true_label_train, curr_tr
    end

    train_pred = [];
    val_pred = [];
    %for model_idx = st:en
    model_idx = 1;
        model_file = strcat(model_prefix, int2str(m
        caffe('init', model_def_file, model_file);
        caffe('set_mode_gpu');

        % training set: 1 to 22
        pred_label_train = [];
        for folder_idx = 1:22
            fname = strcat('/scail/data/group/cvgl/
            data = h5read(fname, '/data');
            cur_N = size(data,4);
            for i = 1 : cur_N
              input_data = {data(:,:,:,i)};
              scores = caffe('forward', input_data)
              cur_label = scores{1};
              pred_label = [pred_label, [cur_label
            end
        end
        train_pred = cat(3, train_pred, pred_label

        % validataion set: 23:24
        pred_label_val = [];
        for folder_idx = 23:24
            fname = strcat('/scail/data/group/cvgl/
            data = h5read(fname, '/data');
            cur_N = size(data,4);
            pred_label = [];
            for i = 1 : cur_N
              input_data = {data(:,:,:,i)};
              scores = caffe('forward', input_data)
              cur_label = scores{1}
              pred_label_val = [pred_label_val, [cu
            end
        end
        val_pred = cat(3, val_pred, pred_label_val)
    %end

save reg_005_GPU.mat train_pred true_label_train v
```

imagenet caffe definition

```
name: "CaffeNet"
layers {
```

```
  name: "data"
  type: HDF5_DATA
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "/scail/data/group/cvgl/u/zhianghu/head_pose2data_227/depth_data_list.txt"
    batch_size: 140
  }
  include: { phase: TRAIN }
}
layers {
  name: "data"
  type: HDF5_DATA
  top: "data"
  top: "label"
  hdf5_data_param {
    source: "/scail/data/group/cvgl/u/zhianghu/head_pose2data_227/depth_data_test_list.txt"
    batch_size: 140
  }
  include: { phase: TEST }
}
layers {
  name: "conv1"
  type: CONVOLUTION
  bottom: "data"
  top: "conv1"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  convolution_param {
    num_output: 96
    kernel_size: 11
    stride: 4
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layers {
  name: "relu1"
  type: RELU
  bottom: "conv1"
  top: "conv1"
}
layers {
  name: "pool1"
  type: POOLING
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "norm1"
  type: LRN
  bottom: "pool1"
  top: "norm1"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layers {
  name: "conv2"
  type: CONVOLUTION
  bottom: "norm1"
  top: "conv2"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  convolution_param {
    num_output: 256
    pad: 2
    kernel_size: 5
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layers {
  name: "relu2"
  type: RELU
  bottom: "conv2"
  top: "conv2"
}
layers {
  name: "pool2"
  type: POOLING
  bottom: "conv2"
```

```
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "norm2"
  type: LRN
  bottom: "pool2"
  top: "norm2"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layers {
  name: "conv3"
  type: CONVOLUTION
  bottom: "norm2"
  top: "conv3"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layers {
  name: "relu3"
  type: RELU
  bottom: "conv3"
  top: "conv3"
}
layers {
  name: "conv4"
  type: CONVOLUTION
  bottom: "conv3"
  top: "conv4"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  convolution_param {
    num_output: 384
    pad: 1
    kernel_size: 3
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layers {
  name: "relu4"
  type: RELU
  bottom: "conv4"
  top: "conv4"
}
layers {
  name: "conv5"
  type: CONVOLUTION
  bottom: "conv4"
  top: "conv5"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
    group: 2
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layers {
  name: "relu5"
  type: RELU
  bottom: "conv5"
  top: "conv5"
```

```
}
layers {
  name: "pool5"
  type: POOLING
  bottom: "conv5"
  top: "pool5"
  pooling_param {
    pool: MAX
    kernel_size: 3
    stride: 2
  }
}
layers {
  name: "fc6"
  type: INNER_PRODUCT
  bottom: "pool5"
  top: "fc6"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 4096
    weight_filler {
      type: "gaussian"
      std: 0.005
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layers {
  name: "relu6"
  type: RELU
  bottom: "fc6"
  top: "fc6"
}
layers {
  name: "drop6"
  type: DROPOUT
  bottom: "fc6"
  top: "fc6"
  dropout_param {
    dropout_ratio: 0.5
  }
}
layers {
  name: "fc7"
  type: INNER_PRODUCT
  bottom: "fc6"
  top: "fc7"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 4096
    weight_filler {
      type: "gaussian"
      std: 0.005
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layers {
  name: "relu7"
  type: RELU
  bottom: "fc7"
  top: "fc7"
}
layers {
  name: "drop7"
  type: DROPOUT
  bottom: "fc7"
  top: "fc7"
  dropout_param {
    dropout_ratio: 0.5
  }
}
layers {
  name: "fc8_ford"
  type: INNER_PRODUCT
  bottom: "fc7"
  top: "fc8_ford"
  blobs_lr: 1
  blobs_lr: 2
  weight_decay: 1
  weight_decay: 0
  inner_product_param {
    num_output: 3
    weight_filler {
      type: "gaussian"
      std: 0.01
    }
    bias_filler {
      type: "constant"
      value: 0
    }
  }
}
layers {
```

```
  name: "accuracy"
  type: EUCLIDEAN_LOSS
  bottom: "fc8_ford"
  bottom: "label"
  top: "accuracy"
  include: { phase: TEST }
}
layers {
  name: "loss"
  type: EUCLIDEAN_LOSS
  bottom: "fc8_ford"
  bottom: "label"
  top: "loss"
}
```