# Head Pose Estimation - Final Report

**Team 10**[1]: Amrutha Varshini Ramesh, Lakshmi Sridevi, Sanket Waghmare

## 1. Problem Statement

In the past few years, research on head pose estimation has increased drastically considering its demand in various Human Machine interaction platforms. One of the main reasons for this increased interest is the various details embodied in the movement of head as a form of gesturing in conversations and other daily activities. For instance, people nod to indicate that they understand something that is being said, they use additional gestures to indicate dissent, confusion, consideration, and agreement. Apart from the regular gesture recognition, there is much that can be inferred by observing a persons head - a quick head movement may be a sign of surprise or alarm. For the above mentioned and many other reasons, fast and robust algorithms for head pose estimation is essential for many face applications like gaze detection, estimating a persons visual attention among others, which are very useful for higher level applications such as driver assistance, augmented reality, comprehending non-verbal communication in human computer interfaces and the like. For example, in the context of autonomous cars, a driving assistance system could take advantage of head pose estimation for decelerating the car when pedestrians do not notice the presence of the vehicle.[2].

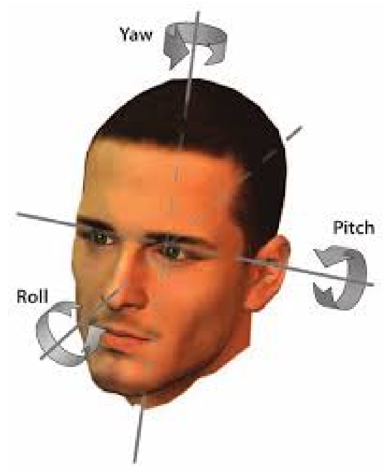
In this project, we implement and improve the proposed robust and efficient algorithm[6] for head pose estimation, which during inference, will identify the pan and tilt position of the head in an unseen image and classifies the image into one of the following 9 classes:

- Left Up (**LU**), Left Center (**LC**), Left Down (**LD**)
- Right UP (**RU**), Right Center(**RC**), Right Down(**RD**)
- Center Up (**CU**), Center(**CC**), Center Down(**CD**)

## 2. Introduction

Human head movement can be characterized by 3 movements: pitch, yaw and roll. The **pitch(tilt)** is the Up-Down movement, **yaw(pan)** is the Left-Right movement and **Roll** is the side-to-side movement. To quantify the position of head, one should measure the above mentioned 3 angles of head with respect to a reference position. For most real world applications, the tilt and pan information of the head position would be sufficient, as it gives most information on the head position like where the person is looking. The side-to-side movement of the head does not interfere much with the person's gaze.[4]

[1]All authors have equal contribution



Most papers attempting to solve this problem frames it as a regression problem, where the pitch and roll angles are estimated directly which thereby allows more flexibility in tuning the tolerance level. But generating data for the regression problem is indeed difficult because measuring the pitch and yaw angles of human head is not natural and not an easy task.

In this project, we are attempting to solve this problem of head pose estimation, wherein we take a head image as input and identify the tilt and pan positions - (up/down/center) for each position. Unlike the regression problem, for this classification problem, generating dataset is easier as it is natural for human beings to identify the head position.

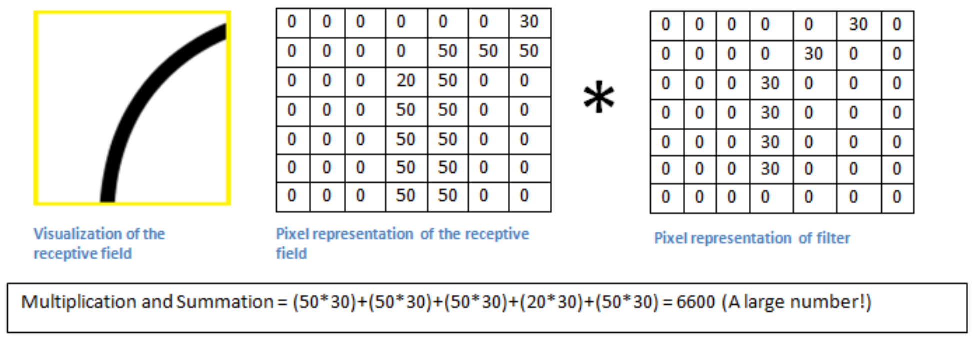
## 3. Preliminaries

To obtain a clear understanding of our algorithm, this section explains the theory of Feature extraction, Neural networks and Convolutional neural network model that are used in our algorithm.

### *3.1. Feature extraction*

Human beings tend to identify known objects by recognizing features and associates it with the object seen before. Here ”features” correspond to important characteristics of the object. For example, human face is identified as one with features such as eyes, nose, lips, eyebrows, etc.

A similar approach of feature extraction is used to help the computers to understand more about images. In the image processing community, a popular method to extract different features from images is to design a filter, which when convolved with the image will output a feature of the image. An example of a simple curve detector is shown below[1]


Visualization of the receptive field
Pixel representation of the receptive field
Pixel representation of filter

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 30 |
| 0 | 0 | 0 | 0 | 50 | 50 | 50 |
| 0 | 0 | 0 | 20 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |
| 0 | 0 | 0 | 50 | 50 | 0 | 0 |

$*$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 30 | 0 |
| 0 | 0 | 0 | 0 | 30 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 30 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

In the above example, ”receptive field” is the image on which the filter is convolved. The curve is the feature. If this feature is present in the receptive field, then convolving the receptive field with the curve detector filter will give a large value. Filters are usually much smaller in size compared to the receptive field itself. So here, the filter is convolved with a small window in the receptive field to identify the feature in the small window. This step is repeated through all the small windows in the receptive field. The output of this feature extraction process through convolution operation($Features_{(W_{out} \times H_{out})}$) will be a matrix called ”Feature map” of size

$$W_{out} = \frac{W_{in} - F_w + 2P}{S_w} + 1 \tag{1}$$

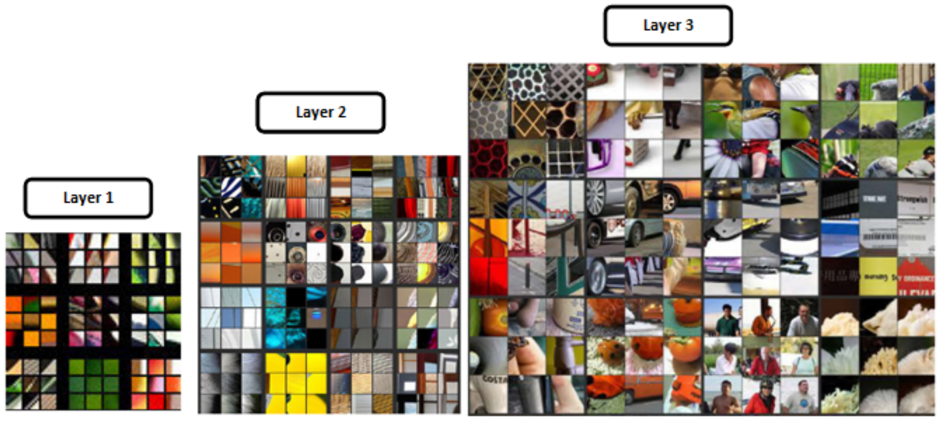
$$H_{out} = \frac{H_{in} - F_h + 2P}{S_h} + 1 \tag{2}$$

where,

- $W_{in}$ and $H_{in}$ are the width and height of the receptive field.
- $F_w$ and $F_h$ are the width and height of the filter
- $P$ is the padding size if any
- $S_w$ and $S_h$ are the stride(number of pixels by which the filter shifts) width and height.

Note that every value in the feature map is nothing but the feature information of the corresponding window in the receptive field. When the convolution with filter is repeated multiple times, every value in the final matrix will provide feature information of a much larger area of the original image. The above mentioned example is for a grayscale 2-D image. For a color image with 3 channels(depth), R, G and B, the filter should also have same depth in order to perform the operation.

The diagram below provides a visual representation of how low level features to higher level features are extracted from the input when the convolution operation is performed multiple times(in different layers).[7]



*Feature Visualization of Convolutional net trained on ImageNet (from Matthew D. Zeiler and Rob Fergus)*

### *3.2. Neural Networks*

In section 3.1, we understood feature extraction through a simple example of curve detection. For much more involved computer vision problems like Head Pose Estimation, complicated features are necessary to successfully capture all the patterns in the image. Consequently, many complicated filters need to be designed to perform this task. This can be time-consuming and potentially impossible for difficult tasks such as Head Pose Estimation.

Neural networks are mathematical models, that have the capability to learn such features automatically in a data-driven fashion. Neural networks are parametrized models that are ”trained” to learn patterns in the data as features. The parameters(weights) of interest are learned from a given dataset, to perform any downstream tasks such as prediction, estimation, classification etc. There are several neural network models, each specializing in one or more of the above mentioned tasks. One such model is the **Convolutional Neural Network(CNN)**, which has the capability to learn the filter(weights) from the dataset. Next section explains the general architecture of a CNN model, which will be helpful to understand the CNN based model used to solve our problem.

### *3.3. Convolutional Neural Networks(CNN)*

CNN models are made up of four main layers which are stacked together.[3]

- Convolutional layer:

  Convolutional layer is the first layer of the CNN model. This layer convolves small learn-able filters(weights) with the input image and outputs a feature map for every channel which are stacked along the dimension of depth.

- Non-linear activation:

  This layer provides non-linearlity in the network. This layer takes in the input from the previous layers and applies a differentiable non-linear function to it. Intuitively, this layer performs non-linear transformation on the input and thus expresses the input in a different space, where the classification task can be easier(input might become linearly seperable).

- Pooling layer:

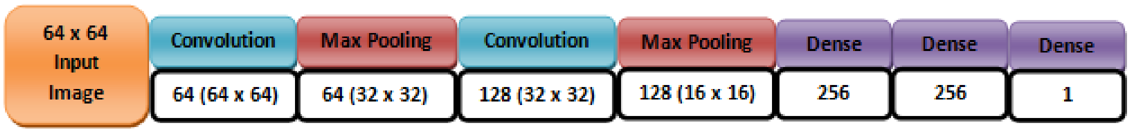
  Pooling layer is inserted periodically into the CNN architecture. The main aim of the pooling layer is to reduce the spatial dimensions of the input to this layer, thereby reducing the number of parameters and computations in the network. This layer receives input from the other layers of the network and outputs a multi-channel 2-D tensor with reduced size, thus retaining only the important information.

- Fully Connected layer:

  The output from the previous convolutional layers and pooling layers represent the high level features of the input image. The fully connected layer utilizes these features to perform the classification task. Thus the fully connected layer, also known as the dense layer, is usually the final layer of the network. This layer takes in the 2-D tensor input and usually outputs a 1-D vector of size equal to number of classes.

Apart from the 4 main above mentioned layers, there are few other layers that can be present in the CNN architecture like dropout layer where random points in the input are dropped and normalization layer which performs normalization of the input.

The diagram below is an example of how a CNN model can look like



CNN Architecture

## *3.4. Neural network training*

This section explains how any neural network model learns its weights from the given dataset. The learning(training) in the neural network model happens in 3 stages

- Forward pass:

  This stage takes in training dataset as the input($X$) and passes the input through the neural network model. The output($\widehat{Y}$) of this stage is the output from the fully connected layer(last layer) of the network(class prediction). The weights are initialized randomly the first time and thus the class prediction will be very far from the true output($Y$) in the beginning. This stage can be represented mathematically as

$$\widehat{Y} = \phi(W.X + b) \tag{3}$$

  where,

  - $X$ is the input(training data)
  - $\widehat{Y}$ is the class prediction
  - $W$ is the weight matrix (filter)
  - $b$ is the bias vector, a small additive component added to the network

  - $\phi$ is the non-linear activation function

- Loss Function:

  Now that the model has predicted an output, $\widehat{Y}$ and we also know the true output for every input in the training dataset $x_i \in X$, we can find the error between true output and the predicted output. The function that computes this error is called loss function. There are several loss functions commonly used by the Machine learning community. The equation below shows the mathematical representation of one such loss function $\mathbb{L}$, called cross-entropy loss. The same is used in our problem, Head Pose estimation as well.

$$\mathbb{L}(Y, \widehat{Y}) = -\sum_i y_i log(\widehat{y_i}) \tag{4}$$

$$\forall\ y_i \in Y \text{ and } \widehat{y_i} \in \widehat{Y}$$

- Backward pass and weight update:

  To achieve accurate classification, the error computed in the equation 4 should be minimal or close to zero. To achieve this, the weights $W$ of the network in equation 3 needs to be adjusted close to the weights, $W^*$ that generated $Y$. Note that we do not know the true weights $W^*$ and learning it from the dataset is the ultimate aim of this learning problem.

$$W = \min_W \ \ \mathbb{L}(Y, \widehat{Y}) \tag{5}$$

  In this stage, the loss that was computed in equation 4 is propagated backwards on the network to understand which weight $w_i \in W$ contributed to the loss and adjust them. This operation is performed by the optimization algorithm, which computes the $W$ that minimizes the loss and updates the weights. There are several optimization algorithms popularly used by the machine learning community. For our problem, we use one such off-the-shelf optimization solver, *Adam Optimizer*.

The above mentioned 3 stages, forward pass, loss computation, backward pass and weight update performed once is called 1 epoch. Several epochs are performed repetitively until the loss value is minimized and converges.

## 4. Implementation

This section explains how the CNN model was developed and trained using the neural network training algorithm explained in section 3.3 and 3.4, to solve the Head Pose estimation problem

### *4.1. Data Generation*

This section explains the file **dataset_generation.py**.

For our implementation, we are using the Prima Head Pose Image dataset.[5] The dataset consists of 2790 monocular face images of 15 subjects with variations in tilt and pan angles in the range [-90,90]. There are 2 series of 93 images with discrete poses for each person. Images in the dataset have occlusions such as glasses and variations such as skin color. Background is willingly neutral and uncluttered in order to focus on face operations. They are generated with face at the center of the image, enabling us to crop the images at constant size to extract faces. The images have file name in the following format:

$$\textbf{"person""person number""image number""tilt angle""pan angle"".jpg"} \tag{6}$$

where,

- person number ranges from 1 to 15
- tilt angle measure the pitch angle, varying from -90 to 90
- pan angles measures yaw, varying from from -90 to 90

We have extracted the true tilt and pan class values from the image file name by the folowing algorithm. Refer 6 for file name pattern.

$$\begin{aligned} if((tilt\ angle >= -90)\ and\ (tilt\ angle <= -16)) &: \ tilt = \ Down\ ;\ tilt\_class = 2 \\ elseif((tilt\ angle >= -15)\ and\ (tilt\ angle <= 15)) &: \ tilt = \ Center\ ;\ tilt\_class = 1 \\ elseif((tilt\ angle >= 16)\ and\ (tilt\ angle <= 90)) &: \ tilt = \ up\ ;\ tilt\_class = 0 \end{aligned} \tag{7}$$

$$\begin{aligned} if((pan\ angle >= -90)\ and\ (pan\ angle <= -16)) &: \ pan = \ Right\ ;\ pan\_class = 2 \\ elseif((pan\ angle >= -15)\ and\ (pan\ angle <= 15)) &: \ pan = \ Center\ ;\ pan\_class = 1 \\ elseif((pan\ angle >= 16)\ and\ (pan\ angle <= 90)) &: \ pan = \ Left\ ;\ pan\_class = 0 \end{aligned} \tag{8}$$

We extract true tilt_class and pan_class $y_i$, for every image in the dataset and store the information in a csv file along with the file name.

Now that the true outputs are extracted, the next step is to generate the input $X$ to the model from the Prima head pose database. The data generation algorithm we implemented is as follows

**Algorithm 1** Data Generation and preprocessing

```
1: procedure DATA GENERATION
2:     dataset ← Prima head pose dataset
3:     $x_i$ ← Every image in dataset
4:     function LOOP($X$)
5:         for $i$ ← 1 to $len(dataset)$ do
6:             $x_i$ ← face extracted by cropping
7:             $x_i$ ← $x_i$ resized to $64 \times 64$                ▷ Save $x_i$
8:             path ← path where $x_i$ is saved
9:             Write path in the csv file
10:    Loop(dataset)
```

## *4.2. Model*

This section explains the file **model_sanket.py**.

In the section 4.1, we have generated and pre-processed inputs $X$ and true class outputs(labels) $Y$. This section explains the architecture of the CNN model we have implemented to solve Head pose estimation problem.

**Algorithm 2** Model architecture

```
 1: procedure LAYER 1(Input dimension: 64x64x1 )
 2:     Convolutional layer                                  ▹ Output: 64x64x64
 3:     tanh non-linear activation layer
 4:     Pooling layer                                        ▹ Output: 32x32x64
 5: procedure LAYER 2(Input dimension: 32x32x64 )
 6:     Convolutional layer                                  ▹ Output: 32x32x128
 7:     tanh non-linear activation layer
 8:     Pooling layer                                        ▹ Output: 16x16x128
 9: procedure LAYER 3(Input dimension: 16x16x128 )
10:     Convolutional layer                                  ▹ Output: 16x16x256
11:     tanh non-linear activation layer
12:     Pooling layer                                        ▹ Output: 8x8x256
13:     Flattening
14: procedure LAYER 4(Input dimension: 16384 )
15:     Fully Connected layer                                ▹ Output: 256
16:     tanh non-linear activation layer
17:     Fully Connected layer                                ▹ Output: 3
18:     tanh non-linear activation layer
19: procedure MODEL OUTPUT
20:     Output:$\widehat{Y}$
```

*4.3. Training*

This section explains the file **training.py and main.py**

The section 3.4, explained the general learning framework. This section describes the training algorithm implemented in out project.

**Algorithm 3** Training

```
1: procedure MODEL TRAINING
2:     $batch\_size = 256$
3:     $W_{old} \leftarrow$ normally distributed weights with $\sigma = 0.5$
4:     function LOOP
5:         for $epoch \leftarrow 1$ to 100 do
6:             for $batch \leftarrow 1$ to 11 do
7:                 Softmax cross entropy loss $\leftarrow Y, network$      ▹ $Y$ is one-hot encoded
8:                 Adam Optimizer $\leftarrow$ Softmax cross entropy loss, $W_{old}$
9:                 $w_{new} \leftarrow$ Adam Optimizer
```

Note that in the training implementation, the true output $Y$ is one-hot encoded to match the dimensions with the model prediction $\widehat{Y}$

### *4.4. Inference*

From the section 4.3, we get the weights $W$ learned from the dataset. Now for any head image, the model with the learned weights $W$ will predict the class label $\widehat{Y}$, that is, estimate the head position during forward pass.

### *4.5. Challenges and Strategy*

This section explains all the challenges we faced during the implementation and the algorithm changes and fixes we gave to overcome the challenge

- **Challenge**: Very high loss
- **Issue**: Small data
- **Solution**: Reduction of number of classes

Initially we designed the model to predict among 25 classes, so that it would provide more precise prediction which is more desirable. But during training, we witnessed that the loss value is too high and was not converging. To analyse the problem and to debug, we tried to predict only 2 classes and it performed very well. Thus we assured that the model was learning but it suffered from insufficient data( only 100 examples per class). So we reduced the number of classes from 25 to 9.

- **Challenge**: Difficulty identifying root cause of high loss
- **Issue**: Complicated problem
- **Solution**: Separate into sub-problems

After reducing the number of classes, the accuracy significantly improved, but it was not good enough. We tried to figure out the main issue that was contributing to the mis-classification. In order to do that, we separated the whole head-pose estimation problem to 2 sub-problems, estimating the tilt position and estimating the pan position. So, currently we train 2 models, one for each head movement, thus reducing the number of classes further to 3. This helped us to identify the right problem that contributed to high loss as well as reduced the number of classes which is desirable.

- **Challenge**: High loss in one particular class
- **Issue**: Very few classes
- **Solution**: Regenerated dataset

Initially, while generating training dataset, we classified the tilt = center, pan = center only if tilt_angle =0 and pan_angle = 0. Even if the head was 15 degrees turned, we classified it as left, right, up or down according to the turn angle. Due to the separation of the main problem into sub-problems, it was easier for us to analyze which class was highly mis-classified.

We found that, in the dataset(of 15 people), for the 15 degrees head turn, some people had their head positioned close to center and for some people head was turned away from the center. We understood that the dataset was generated by marking angle on the wall and different people were made to see the mark while image was captured. The head turn angle would vary according to their height would not be consistant. To fix this problem, we regenerated the labels with head positioned between -15 and 15 as center. This fix significantly improved our results.

## 5. Evaluation

To evaluate our model, we used the metrics, total accuracy and class accuracy. They can be calculated as

$$\text{Total accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{9}$$

where,

- Number of correct predictions : Total number of times when $Y = \widehat{Y}$
- Total number of predictions : Batch size * number of batches

Total accuracy shows the performance of the model. If the total accuracy is high, it means that the model is classifying most examples correctly. If the total accuracy is <0.5, it means that the model has not learned anything from the dataset and it is predicting randomly.

The class accuracy for every class, was calculated as

$$\text{Class accuracy} = \frac{\text{Number of correct class predictions}}{\text{Total number of true occurrence of the class}} \tag{10}$$

This metric gives a good understanding of which class is contributing more to the total accuracy and which is reducing the accuracy and in general how the classification works for every class.

## 6. Results

The metrics explained in 5 were used to compute the results shown below. The results were calculated on the training data. The test accuracy was not calculated. Splitting the dataset into train and test further reduced the dataset size causing reduction in training accuracy.

### *6.1. Total Accuracy*

The table below gives the total accuracy for the tilt and pan models

Table 1: **Total accuracy : Tilt**

| Epoch | Accuracy |
|---|---|
| 10 | 0.158401489258 |
| 20 | 0.69921875 |
| 30 | 0.715087890625 |
| 40 | 0.75 |
| 50 | 0.82127837 |

The accuracy converges after 50 epochs to around **83%** for tilt.

Table 2: **Total accuracy : Pan**

| Epoch | Accuracy |
|---|---|
| 10 | 0.461090087891 |
| 20 | 549468994141 |
| 30 | 0.520782470703 |
| 40 | 0.531768798828 |
| 50 | 0.526611328125 |
| 60 | 0.575256347656 |
| 115 | 0.685487 |

The accuracy converges after 115 epochs to approximately **70%** for pan.

### *6.2. Class accuracy:*

The table below gives the class accuracy for the tilt and pan models.

Table 3: **Class accuracy : Tilt**

| Epoch | Up | Center | Down |
|---|---|---|---|
| 10 | 0.84375 | 0.29296875 | 0.86328125 |
| 20 | 0.84765625 | 0.3125 | 0.83984375 |
| 30 | 0.8671875 | 0.2734375 | 0.859375 |
| 40 | 0.9140625 | 0.25 | 0.8359375 |
| 50 | 0.8950224 | 0.33984375 | 0.875 |

Table 4: **Class accuracy : Pan**

| Epoch | Left | Center | Right |
|---|---|---|---|
| 10 | 0.75390625 | 0.453125 | 0.79296875 |
| 20 | 0.7734375 | 0.44140625 | 0.78515625 |
| 30 | 0.7578125 | 0.46875 | 0.7734375 |
| 40 | 0.7539062 | 0.46484375 | 0.78125 |
| 50 | 0.75390625 | 0.468755 | 0.77734375 |
| 60 | 0.8125 | 0.421875 | 0.765625 |
| 115 | 0.834897 | 0.4732 | 0.819067892 |

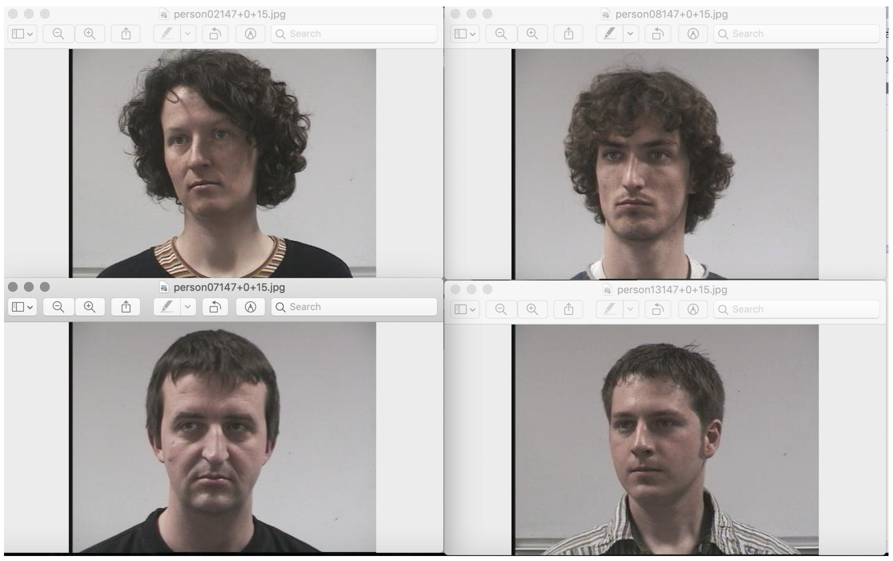
The class accuracy tables for both tilt and pan clearly shows that the classification for center is the major contributor in reducing the total accuracy. In both the cases, tilt and pan, the model does not classify the center images correctly and the mis-classifications are split between the other two classes. This explains the lower accuracy in the non-center classes too.

## 7. Results Analysis and Possible fixes

The section 6 showed that the ”center” class in both the models were the major contributor to the mis-classifications. We have analyzed the possible reasons and also have suggested possible fixes in this section.

### 7.1. *Analysis*

- In the section 4.5, we had explained the regeneration of the dataset to modify the ”center” class in order to improve accuracy. It was a trick to improve accuracy, but it did not solve the actual inconsistency in the dataset.
- The actual problem is that for certain tilt_angle and pan_angle, few images in the dataset look close to center and few images look non-center. These cases are hard even for humans to classify. An example to portray this problem is shown below.



They are the images of 4 different people turning 15 degrees left. We could clearly see that, the head image in the 2nd row 1st column appears to be positioned towards center whereas the other images seem to have positioned towards left. Note that in our training dataset we classify all these images as center.

- The other problem is, the number of images for the center class is significantly less comparing other classes. We classify the images with angles between -15 and +15 as center and angles between ±16 and ±90 as up/down/left/right class. Therefore the other classes have double the amount of training examples than center class.
- To improve the model performance with same dataset, we could try approaches like data augmentation, that is, tricking the model by presenting the transformed images of the the existing images, as new images. The transformations like blur, adding noise, scaling can be applied to the existing images to generate more images.
- Another approach is to use much powerful models. We use a basic CNN model in this project. We could try using proven powerful models like ResNet, AlexNet etc.

### 7.2. *Proposed fixes*

- The ideal solution to the problem explained in section 7.1 is generating more data.

- Our approach to this Head pose estimation problem provides more flexibility in annotating the class label for every image to generate training data, as it is a natural skill for human beings to perform this classification. Therefore, more data with consistant dataset would improve performance of the model.

## 8. References

[1] Adit Deshpande. A-beginner's-guide-to-understanding-convolutional-neural-networks-part-2.

[2] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf. Survey of pedestrian detection for advanced driver assistance systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1239–1258, July 2010.

[3] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition.

[4] E. Murphy-Chutorian and M. M. Trivedi. Head pose estimation in computer vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):607–626, April 2009.

[5] J. L. Crowley N. Gourier, D. Hall. Estimating face orientation from robust detection of salient facial features. *Proceedings of Pointing 2004, ICPR,International Workshop on Visual Observation of Deictic Gestures, Cambridge, UK*.

[6] Massimiliano Patacchiola and Angelo Cangelosi. Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods. *Pattern Recognition*, 71:132 – 143, 2017.

[7] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *European Conference on Computer Vision*, pages 818–833, 2014.