CS771 Project Report

Facial Keypoints Detection

by

GROUP - 21 (Rahul Meena, Salman Thind, Shreyan Chowdhury, Sunil Kumar)

April 2015

Abstract

This project investigates several methods for detection of keypoints in images of faces. Keypoints include eye centers, nose tip, lip centers, mouth corners etc. Methods that have been analyzed and implemented include patch correlation, neural network and convolutional neural network. In this report, we present detailed analysis along with results for each of these methods. We also elaborate on dataset filtering to remove outlier images and other manipulation that was done in order to further reduce error rates. We go on to show the scheme that resulted in the best RMSE value - convolutional neural network with split datasets.

Acknowledgements

We wish to thank Prof. Harish Karnick for being a patient and immaculate instructor and guide throughout the course, and for giving us the opportunity to field-test the knowledge gained in his class on a real world problem. We are also extremely grateful to the course tutors and TAs, especially Pranjal Singh, for helping out in times of need.

Contents

Chapter 1

Introduction

The problem is taken from kaggle.com (http://www.kaggle.com/c/facial-keypoints-detection). The objective of this task is to predict keypoint positions on face images. This can be used as a building block in several applications, such as:

- tracking faces in images and video
- analysing facial expressions
- detecting dysmorphic facial signs for medical diagnosis
- biometrics / face recognition

Detecting facial keypoints is a very challenging problem. Facial features vary greatly from one individual to another, and even for a single individual, there is a large amount of variation due to 3D pose, size, position, viewing angle, and illumination conditions. Computer vision research has come a long way in addressing these difficulties, but there remain many opportunities for improvement.

1.1 Problem statement

Given an image of size 96x96 pixels, output the x and y coordinates of facial keypoints, specifically:

1. Left eye center
2. Right eye center
3. Left eye inner corner

4. Right eye inner corner
5. Left eye outer corner
6. Right eye outer corner
7. Left eyebrow inner end
8. Right eyebrow inner end
9. Left eyebrow outer end
10. Right eyebrow outer end
11. Nose tip
12. Mouth left corner
13. Mouth Right corner
14. Mouth center top lip
15. Mouth center bottom lip

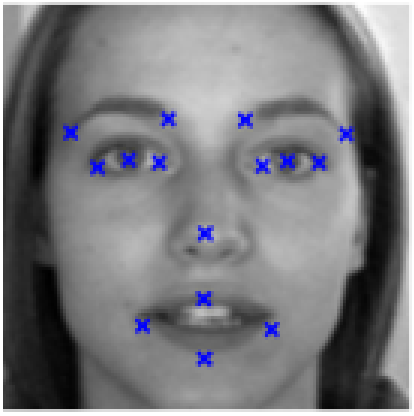

FIGURE 1.1: Example of a training image with 15 keypoints marked

1.2 Evaluation

Submissions are scored on the root mean squared error. RMSE is very common and is a suitable general-purpose error metric. Compared to the Mean Absolute Error, RMSE punishes large errors:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1.1}$$

where y hat is the predicted value and y is the original value.

1.3 Dataset Summary

Each predicted keypoint is specified by an (x,y) real-valued pair in the space of pixel indices. There are 15 keypoints, which represent the above mentioned facial features.

(Left and right here refers to the point of view of the subject.)

In some examples, some of the target keypoint positions are missing (encoded as missing entries in the csv, i.e., with nothing between two commas).

The input image is given in the last field of the data files, and consists of a list of pixels (ordered by row), as integers in (0,255). The images are 96x96 pixels.

Data Files

- **training.csv**: list of training 7049 images. Each row contains the (x,y) coordinates for 15 keypoints, and image data as row-ordered list of pixels.
- **test.csv**: list of 1783 test images. Each row contains ImageId and image data as row-ordered list of pixels
- **submissionFileFormat.csv**: list of 27124 keypoints to predict. Each row contains a RowId, ImageId, FeatureName, Location. FeatureName are "left_eye_center_x," "right_eyebrow_outer_end_y," etc. Location is what you need to predict.

Chapter 2

Dataset Analysis and Modificaion

Detecting facial keypoints is a challenging problem distortions in image (such as rotation, or different orientations of the face) can make it a tough machine learning task.

Dataset summary, tabulated:

Training Set Size	7044
Image Size	96x96
Color	Greyscale
Number of Keypoints	15
Number of Attributes	30 (15x2, x and y coordinates)
Test Set Size	1783

Table 2.1: Dataset Summary

Each image has 9216 pixels. The total number of values in the training vector is thus 64917504, which is a huge number in itself. Also, the input dimension is greater than the number of training examples.

These facts point towards careful analysis of the dataset, and if required, towards necessary modifications to be done for better performance of algorithms intended to be used. A look at Table 2.1 gives a clear picture of the dataset at a glance.

2.1 Splitting the Dataset

An interesting twist with the dataset is that for some of the keypoints we only have about 2,000 labels, while other keypoints have more than 7,000 labels available for training.

The following function was written to print out how many training labels are available for each of the keypoints:

```
if cols:  # get a subset of columns
        df = df[list(cols) + ['Image']]
print(df.count())
```

Output:

```
left_eye_center_x            7034
left_eye_center_y            7034
right_eye_center_x           7032
right_eye_center_y           7032
left_eye_inner_corner_x      2266
left_eye_inner_corner_y      2266
left_eye_outer_corner_x      2263
left_eye_outer_corner_y      2263
right_eye_inner_corner_x     2264
right_eye_inner_corner_y     2264
...
mouth_right_corner_x         2267
mouth_right_corner_y         2267
mouth_center_top_lip_x       2272
mouth_center_top_lip_y       2272
mouth_center_bottom_lip_x    7014
mouth_center_bottom_lip_y    7014
```

It can be seen that for some keypoints, there are only around 2200 labels, and for some, there are more than 7000. Quite evidently, we cannot train a machine learning algorithm with the dataset in its entirety, as there are too many images with missing values.

It was noticed that there are two sets of images in the dataset - one having 15 keypoints, and the other having only 4. Based on whether or not there are missing values in the training observations, the images were segregated and saved as different datasets.

2.2 Outliers

We noted that not all images had a clear full view of a face. Some images had a small face, while some had such a close up shot that parts of the face were outside the frame. These images had to be taken as outliers and removed from the dataset as they would not contribute to learning.

The metric of detecting outliers was chosen as separation between eye centers. The following box plot shows the outliers as dots.

Eye center separation was chosen because it is a good measure of zoom. Faces too big or too small in the image add to inaccuracies in training.

The outliers based on this metric (271 in number) were removed from the dataset.

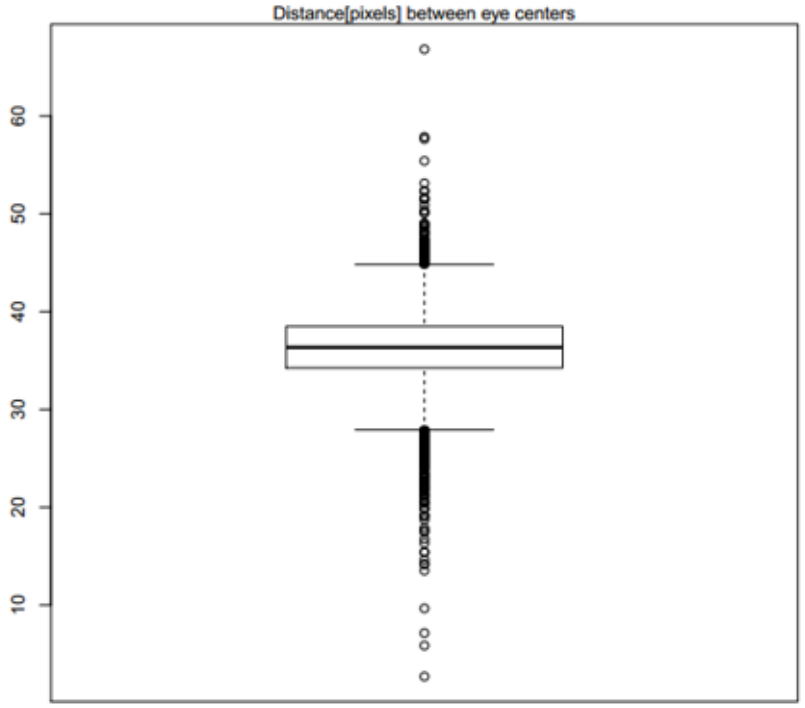

FIGURE 2.1: Box plot for outlier images

FIGURE 2.2: Outlier image examples - too zoomed out

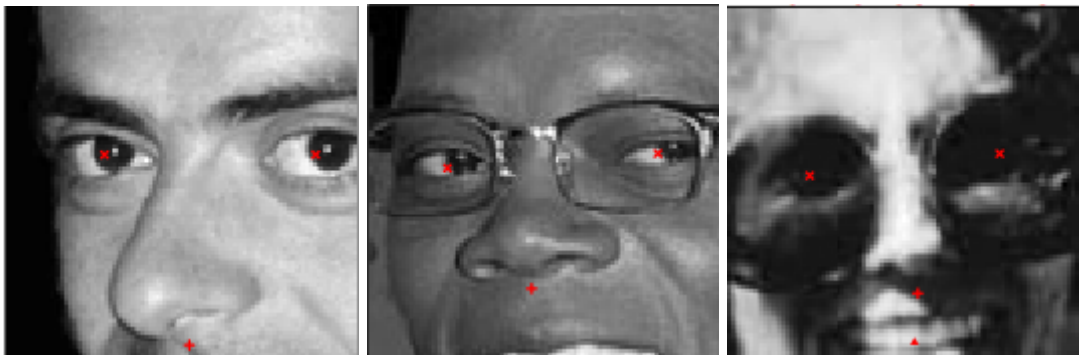

FIGURE 2.3: Outlier image examples - too zoomed in

2.3 Data Augmentation

Data augmentation lets us artificially increase the number of training examples by applying transformations, adding noise etc. That's obviously more economic than having to go out and collect more examples by hand. Augmentation is a very useful tool to have in a neural network or deep learning framework. An overfitting net can generally be made to perform better by using more training data.

We have performed data augmentation by producing mirror images of images and treating them as new data images.

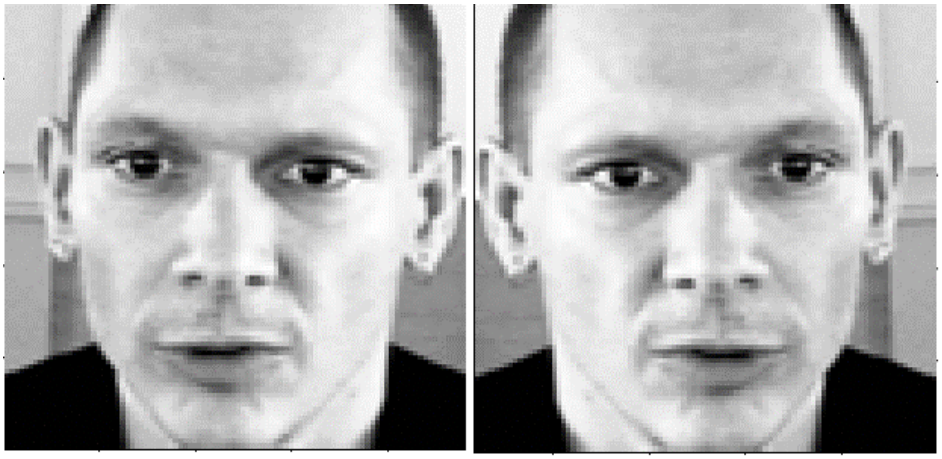

FIGURE 2.4: Example of a mirrored image

2.4 Image Equalization/Normalization

Finally, all the images were equalized such that the brightest pixel value is 255 and the darkest pixel value is 0. This was an essential step before prediction using neural networks, as neural networks are sensitive to individual pixel values.

Chapter 3

Training and Prediction

In this chapter we take a look at the different methods that were investigated for prediction of keypoints, their underlying theory in brief, and the results that were obtained by us. The first sections explains the preliminary methods, and the subsequent sections deal with more elaborate techniques of neural network and convolutional neural network.

3.1 Preliminary methods

3.1.1 Average value

One of the simplest things to try is to compute the mean of the coordinates of each keypoint in the training set and use that as a prediction for all images. This is a very simplistic algorithm, as it completely ignores the images. Computing the mean for each column is straightforward and we get the following result:

left_eye_center_x	left_eye_center_y	right_eye_center_x
66.35902	37.65123	30.30610
right_eye_center_y	left_eye_inner_corner_x	left_eye_inner_corner_y
37.97694	59.15934	37.94475
left_eye_outer_corner_x	left_eye_outer_corner_y	right_eye_inner_corner_x
73.33048	37.70701	36.65261
right_eye_inner_corner_y	right_eye_outer_corner_x	right_eye_outer_corner_y
37.98990	22.38450	38.03350
left_eyebrow_inner_end_x	left_eyebrow_inner_end_y	left_eyebrow_outer_end_x
56.06851	29.33268	79.48283
left_eyebrow_outer_end_y	right_eyebrow_inner_end_x	right_eyebrow_inner_end_y
29.73486	39.32214	29.50300
right_eyebrow_outer_end_x	right_eyebrow_outer_end_y	nose_tip_x
15.87118	30.42817	48.37419
nose_tip_y	mouth_left_corner_x	mouth_left_corner_y
62.71588	63.28574	75.97071
mouth_right_corner_x	mouth_right_corner_y	mouth_center_top_lip_x

```
                32.90040                  76.17977                  47.97541
mouth_center_top_lip_y mouth_center_bottom_lip_x mouth_center_bottom_lip_y
                72.91944                  48.56947                  78.97015
```

Keypoints on test images were predicted with these values, and an RMSE value of **3.96244** was obtained.

3.1.2 Using image patches

The above method was rather simplistic, and did not analyse the images at all. Said another way, we did not use the information about the intensity of each pixel to identify the keypoints.

Therefore, as a next step, taking the average value of a neighbourhood of each keypoint was attempted. The predicted value for a test image would be that for which the correlation with this *average patch* is maximum.

A 21x21 patch around each keypoint pixel was taken and averaged over all training images. Some of the resulting average patches are shown below:

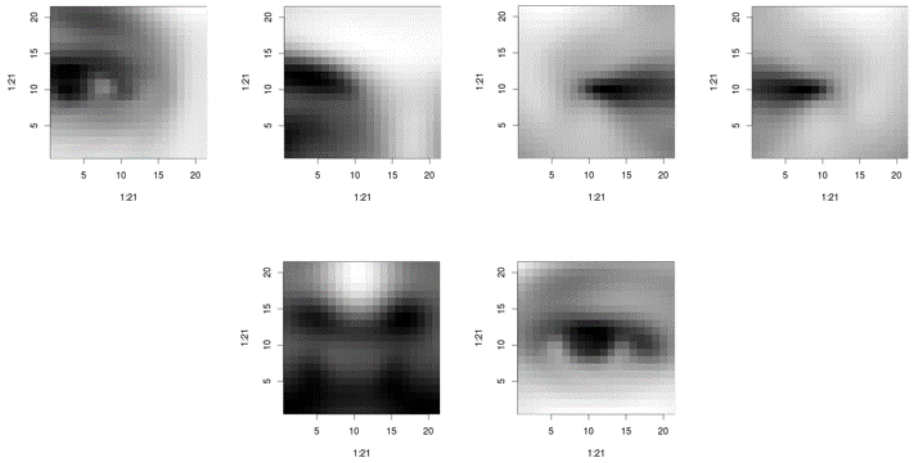

FIGURE 3.1: Average patches for a few keypoints

Now, for prediction, given a test image we need to try all translations, and see which one best matches the average patch. We do that by taking patches of the test images around these points and measuring their correlation with the average patch. This was done in R language:

```
im <- matrix(data = im.test[1,], nrow=96, ncol=96)

r  <- foreach(j = 1:nrow(params), .combine=rbind) %dopar% {
    x     <- params$x[j]
```

```
    y     <- params$y[j]
    p     <- im[(x-patch_size):(x+patch_size), (y-patch_size):(y+patch_size)]
    score <- cor(as.vector(p), as.vector(mean.patch))
    score <- ifelse(is.na(score), 0, score)
    data.frame(x, y, score)
}
best <- r[which.max(r$score), c("x", "y")]
best

    x  y
22 65 39
```

Doing this for all test images, and submitting the result, we achieved an RMSE value of **3.80461**.

3.1.3 Haar-like features

Haar-like features are digital image features used in object recognition. In the detection phase of the ViolaJones object detection framework, a window of the target size is moved over the input image, and for each subsection of the image the Haar-like feature is calculated. This difference is then compared to a learned threshold that separates non-objects from objects. Because such a Haar-like feature is only a weak learner or classifier (its detection quality is slightly better than random guessing) a large number of Haar-like features are necessary to describe an object with sufficient accuracy. In the ViolaJones object detection framework, the Haar-like features are therefore organized in something called a classifier cascade to form a strong learner or classifier.

In our prediction, we used open source Haar classifiers for eyes, nose and mouth in order to give us the locations of these objects. However, predicting the exact coordinate of the required keypoints was not possible with this method and hence it was discontinued.

3.2 Neural network

Object recognition from images immediately points to neural networks as a possible machine learning tool to be used. Various neural network architectures were tried. The number of neurons in the hidden layers were chosen on the basis of a trade-off between training time and prediction accuracy.

3.2.1 Single layer architecture 9216 - 100 - 30

A neural network was trained in the R language using the neuralnet package. The original image vectors of size 9216 (96x96) were taken for training the single layer with 100 neurons neural network. The output layer consisted of 30 outputs, one for each attribute.

```
library(neuralnet)
fmla <- as.formula(paste("train[,9217]~",
        paste0(paste(paste0("train[,",1:9216),collapse="]+"),"]")))
nn <- neuralnet(fmla,train,hidden = 100, threshold = 0.01)
net.results <- compute(nn, test)
print(net.results)
```

With this network, we obtained an RMSE of **3.78336**. Also, this network took a long time to train, and hence tinkering with it was not feasible given the time constraints. We thus, found a C package to help us train the networks faster. The following subsection elaborates on this.

3.2.2 Double layer architecture with decimated input (2304 - 30 - 30 - 30)

A fast neural network C package (FANN, Fast Artificial Neural Network) was used to train this architecture. The input was taken to be 48x48 images, and the network had two hidden layers of 30 neurons each and an output layer of 30 outputs. Activation function was the standard sigmoid function. A plot of epoch vs. error rate for this network is given below:

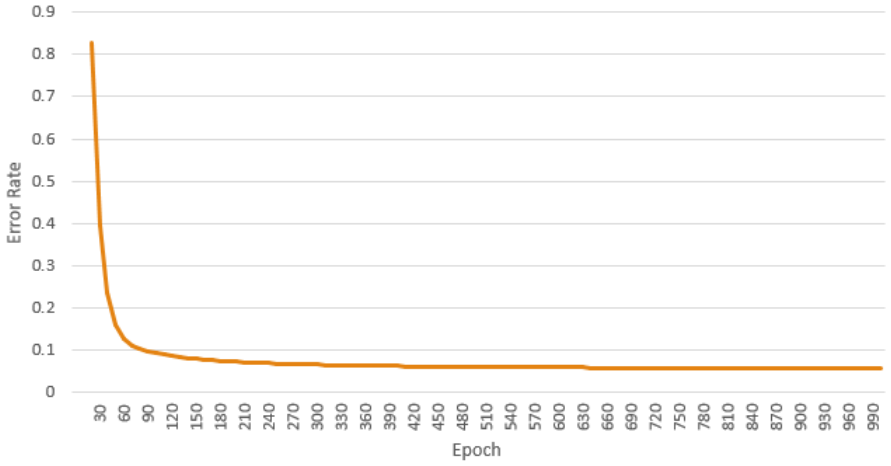

FIGURE 3.2: NN Training

We achieved an error rate of **3.42526** with this network trained on the whole training set and tested on the whole test set.

It was conjectured that training and testing on the previously split datasets [Chap 2] could give better results. We thus went ahead and trained the network separately on the two datasets and then performed prediction, resulting in interesting results.

Training Dataset	Testing Dataset	RMSE
Whole	Whole	3.42526
15-kps 4-kps	15-kps 4-kps	3.42540
15-kps Whole	15-kps 4-kps	3.09656

TABLE 3.1: NN Results

Here, 15-kps refers to the dataset with 15 keypoints, and 4-kps refers to the dataset with 4 keypoints.

3.2.3 Using augmented datasets

When the network was trained with the augmented dataset, surprisingly, the error increased. We obtained an RMSE of **3.46324** with that network, which is greater than what we obtained with the normal dataset. We suspect that this happened because of overfitting.

3.3 Convolutional neural networks

Convolutional layers are different to fully connected layers; they use a few tricks to reduce the number of parameters that need to be learned, while retaining high expressiveness. These are:

- local connectivity: neurons are connected only to a subset of neurons in the previous layer,
- weight sharing: weights are shared between a subset of neurons in the convolutional layer (these neurons form what's called a feature map),
- pooling: static subsampling of inputs.

Units in a convolutional layer actually connect to a 2-d patch of neurons in the previous layer, a prior that lets them exploit the 2-d structure in the input.

We built a convolutional neural net with three convolutional layers and two fully connected layers. Each conv layer is followed by a 2x2 max-pooling layer. Starting with 32 filters, we double the number of filters with every conv layer. The densely connected hidden layers both have 500 units.

This was written in Python using the Lasagne package. GPU was used to speed up training using the Theano package.

```
net2 = NeuralNet(
    layers=[
        ('input', layers.InputLayer),
        ('conv1', layers.Conv2DLayer),
        ('pool1', layers.MaxPool2DLayer),
        ('conv2', layers.Conv2DLayer),
        ('pool2', layers.MaxPool2DLayer),
        ('conv3', layers.Conv2DLayer),
        ('pool3', layers.MaxPool2DLayer),
        ('hidden4', layers.DenseLayer),
        ('hidden5', layers.DenseLayer),
        ('output', layers.DenseLayer),
        ],
    input_shape=(None, 1, 96, 96),
    conv1_num_filters=32, conv1_filter_size=(3, 3), pool1_ds=(2, 2),
    conv2_num_filters=64, conv2_filter_size=(2, 2), pool2_ds=(2, 2),
    conv3_num_filters=128, conv3_filter_size=(2, 2), pool3_ds=(2, 2),
    hidden4_num_units=500, hidden5_num_units=500,
    output_num_units=30, output_nonlinearity=None,

    update_learning_rate=0.01,
    update_momentum=0.9,

    regression=True,
    max_epochs=1000,
    verbose=1,
    )

X, y = load2d()  # load 2-d data
net2.fit(X, y)
```

With this, an RMSE of **3.36401** was obtained, which is better than that obtained from the simple neural network.

With the convolutional neural network trained on whole dataset and tested on 15-kps dataset, and the simple neural network trained on whole dataset and tested on 4-kps dataset, we obtained our best RMSE of **2.95569**

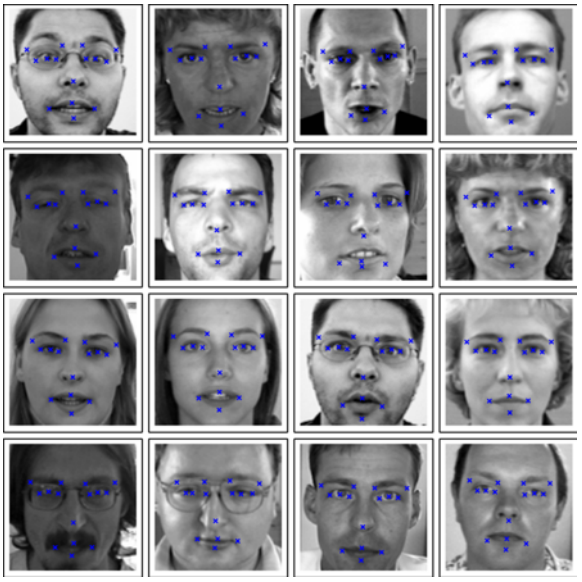

FIGURE 3.3: Some images with overlaid predictions for keypoints

Chapter 4

Results and Discussion

A summary of the results obtained by prediction from different methods is given below.

4.1 Summary of results

Method	Training Dataset	Testing Dataset	RMSE
Average value	Whole	Whole	3.96244
Average patch correlation	Whole	Whole	3.80461
Neural network (2304-30-30-30)	Whole	Whole	3.42526
	15-kps	15-kps	3.42540
	4-kps	4-kps	
	15-kps	15-kps	3.09656
	Whole	4-kps	
Neural network (9216-100-30)	Whole	Whole	3.78336
Convolutional neural network	Whole	Whole	3.36401
	Whole	15-kps	2.95569

Table 4.1: Summary of results

We can see that the best results were obtained from convolutional neural networks. This was expected because convolutional nets take into account the 2D structure of the input space and computes hierarchical features layer by layer. Thus, it is accepted widely that convolutional neural nets work well in vision applications.

We also noticed that data augmentation by taking mirror images did not improve RMSE but rather resulted in it getting worse. We suspect it to be due to overfitting but further investigation could be done on this.

Chapter 5

Further Improvements

This project holds a lot more avenues to be explored and methods to be investigated in order to make the training better and make better predictions on test images. Some of the hings that could be tried after this are given below, as thought out by us:

- **Haar features and neural networks:** A hybrid of Haar detection and neural network can be applied to increase prediction accuracies. Haar feaures to be used for localising the eye/nose/mouth and neural net to be used for prediction of precise coordinates of keypoints.
- **Eigenface approach:** The eigenface approach can be used to detect locations of eye/nose/mouth and thereafter coordinates predicted by correlation or neural network approach.
- **Training CNN with each keypoint separately:** One CNN for each keypoint can be built and trained. This can give better results as each network will be a *specialist* for a keypoint and thus will hopefully give more accurate predictions for that keypoint.

Appendix A

Neural Network Code

```
#include "fann.h"
#include <CL/cl.h>

int main()
{
    const unsigned int max_epochs = 1000;
    const unsigned int epochs_between_reports = 10;

    const unsigned int num_input = 48*48;
    const unsigned int num_output = 30;
    const unsigned int num_layers = 2;
    const unsigned int num_neurons_hidden = 30;

    const float desired_error = (const float) 0.0000;

    fann_type *calc_out;
    unsigned int i;
    int incorrect,ret = 0;
    int orig,pred; float max =0 ;
    float learning_rate = 0.01;

    struct fann *ann = fann_create_standard(num_layers, num_input, num_output);

    fann_set_activation_function_hidden(ann, FANN_SIGMOID);
    fann_set_activation_function_output(ann, FANN_LINEAR);
    fann_set_learning_rate(ann, learning_rate);

    fann_train_on_file(ann, "facial-train.txt", max_epochs,
        epochs_between_reports, desired_error);

    fann_reset_MSE(ann);

    struct fann_train_data *data = fann_read_train_from_file("facial-test.txt");

    printf("Testing network..\n");

    for(i = 0; i < fann_length_train_data(data); i++) {
        calc_out = fann_test(ann, data->input[i], data->output[i] );
```

```
        printf ("%i ", i );
        max = calc_out[0];
        int maxo = data->output[i][0];
        for (int n=0; n<30; n++) {
            printf (" %.2f/%.2f(%.2f) ",calc_out[n]*(2*96), data->output[i][n]*(2*96),
            data->output[i][n]*(2*96) - calc_out[n]*(2*96)  );
        }
        printf ("\n");
    }

    printf("Mean Square Error: %f\n", fann_get_MSE(ann));

    fann_save(ann, "facial.net");

        fann_destroy_train(data);
        fann_destroy(ann);

    return 0;
}
```

Appendix B

Convolutional Neural Network Code

```
import os

import numpy as np
from matplotlib import pyplot
from pandas.io.parsers import read_csv
from sklearn.utils import shuffle
from lasagne import layers
from lasagne.updates import nesterov_momentum
from nolearn.lasagne import NeuralNet

FTRAIN = '/home/sam/fkd/training.csv'
FTEST = '/home/sam/fkd/test.csv'


def load(test=False, cols=None):
        """Loads data from FTEST if *test* is True, otherwise from FTRAIN.
        Pass a list of *cols* if you're only interested in a subset of the
        target columns.
        """
        fname = FTEST if test else FTRAIN
        df = read_csv(os.path.expanduser(fname))  # load pandas dataframe

        # The Image column has pixel values separated by space; convert
        # the values to numpy arrays:
        df['Image'] = df['Image'].apply(lambda im: np.fromstring(im, sep=' '))

        if cols:  # get a subset of columns
                df = df[list(cols) + ['Image']]

        print(df.count())  # prints the number of values for each column
        df = df.dropna()  # drop all rows that have missing values in them

        X = np.vstack(df['Image'].values) / 255.  # scale pixel values to [0, 1]
```

```
        X = X.astype(np.float32)

        if not test:  # only FTRAIN has any target columns
                y = df[df.columns[:-1]].values
                y = (y - 48) / 48  # scale target coordinates to [-1, 1]
                X, y = shuffle(X, y, random_state=42)  # shuffle train data
                y = y.astype(np.float32)
        else:
                y = None
        return X, y

def load2d(test=False, cols=None):
        X, y = load(test=test)
        X = X.reshape(-1, 1, 96, 96)
        return X, y


X, y = load()

net3 = NeuralNet(
        layers=[
                ('input', layers.InputLayer),
                ('conv1', layers.Conv2DLayer),
                ('pool1', layers.MaxPool2DLayer),
                ('conv2', layers.Conv2DLayer),
                ('pool2', layers.MaxPool2DLayer),
                ('conv3', layers.Conv2DLayer),
                ('pool3', layers.MaxPool2DLayer),
                ('hidden4', layers.DenseLayer),
                ('hidden5', layers.DenseLayer),
                ('output', layers.DenseLayer),
                ],
        input_shape=(None, 1, 96, 96),
        conv1_num_filters=32, conv1_filter_size=(3, 3), pool1_ds=(2, 2),
        conv2_num_filters=64, conv2_filter_size=(2, 2), pool2_ds=(2, 2),
        conv3_num_filters=128, conv3_filter_size=(2, 2), pool3_ds=(2, 2),
        hidden4_num_units=500, hidden5_num_units=500,
        output_num_units=30, output_nonlinearity=None,

        update_learning_rate=0.01,
        update_momentum=0.9,

        regression=True,
        #batch_iterator_train=FlipBatchIterator(batch_size=128),
        max_epochs=3000,
        verbose=1,
        )
X, y = load2d();
net3.fit(X, y)

import cPickle as pickle
with open('net3.pickle', 'wb') as f:
    pickle.dump(net3, f, -1)

X, _ = load2d(test=True)
```

```
y_pred = net3.predict(X)

f=open('fff','r+')
snr = 1;
```

Bibliography

[1] Kaggle Page, http://www.kaggle.com/c/facial-keypoints-detection

[2] Haar Training, http://note.sonots.com/SciSoftware/haartraining.html

[3] Viola, Paul, and Michael Jones, *"Robust real-time object detection"* International Journal of Computer Vision 4 (2001): 34-47

[4] R Tutorial, http://www.cyclismo.org/tutorial/R/

[5] Neural Networks in R tutorial, http://gekkoquant.com/2012/05/26/neural-networks-with-r-simple-example/

[6] Fast Artificial Neural Network, http://leenissen.dk/fann/html/files2/gettingstarted-txt.html

[7] Theano for NN training on GPU http://deeplearning.net/software/theano/

[8] Lasagne package for NN in Python http://nbviewer.ipython.org/github/craffel/Lasagne-tutorial/blob/master/examples/tutorial.ipynb