Capstone Project

Aparajit Chatterjee

26-March-2019

Title: Facial Keypoint Detection

Definition

Project Overview

Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images. Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene.

Face detection is used in biometrics, often as a part of (or together with) a facial recognition system. It is also used in video surveillance, human computer interface and image database management.

Problem Statement

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:

1. Tracking faces in images and video
2. Analysing facial expressions
3. Detecting dysmorphic facial signs for medical diagnosis
4. Biometrics / face recognition

Detecting facial key-points 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.

To tackle this problem, I will use three base models:

1. **A fully connected model**
2. **A Simple CNN**
3. **Using Image Augmentation with a CNN**

Metrics

Mean Squared Error (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

In statistics, the mean squared error (MSE) or mean squared deviation (MSD) of an estimator (of a procedure for estimating an unobserved quantity) measures the average of the squares of the errors—that is, the average squared difference between the estimated values and what is estimated. MSE is a risk function, corresponding to the expected value of the squared error loss. The fact that MSE is almost always strictly positive (and not zero) is because of randomness or because the estimator does not account for information that could produce a more accurate estimate.[1]

The MSE is a measure of the quality of an estimator—it is always non-negative, and values closer to zero are better.

Analysis

Data Exploration

The Dataset is collected from kaggle.

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 following elements of the face:

left_eye_center, right_eye_center, left_eye_inner_corner, left_eye_outer_corner, right_eye_inner_corner, right_eye_outer_corner, left_eyebrow_inner_end, left_eyebrow_outer_end, right_eyebrow_inner_end, right_eyebrow_outer_end, nose_tip, mouth_left_corner, mouth_right_corner, mouth_center_top_lip, mouth_center_bottom_lip

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.

Exploratory Visualization

The Dataset looks like this:

	index	0
0	left_eye_center_x	7039
1	left_eye_center_y	7039
2	right_eye_center_x	7036
3	right_eye_center_y	7036
4	left_eye_inner_corner_x	2271
5	left_eye_inner_corner_y	2271
6	left_eye_outer_corner_x	2267
7	left_eye_outer_corner_y	2267
8	right_eye_inner_corner_x	2268
9	right_eye_inner_corner_y	2268
10	right_eye_outer_corner_x	2268
11	right_eye_outer_corner_y	2268
12	left_eyebrow_inner_end_x	2270
13	left_eyebrow_inner_end_y	2270
14	left_eyebrow_outer_end_x	2225
15	left_eyebrow_outer_end_y	2225
16	right_eyebrow_inner_end_x	2270
17	right_eyebrow_inner_end_y	2270
18	right_eyebrow_outer_end_x	2236
19	right_eyebrow_outer_end_y	2236
20	nose_tip_x	7049
21	nose_tip_y	7049
22	mouth_left_corner_x	2269
23	mouth_left_corner_y	2269
24	mouth_right_corner_x	2270
25	mouth_right_corner_y	2270
26	mouth_center_top_lip_x	2275
27	mouth_center_top_lip_y	2275
28	mouth_center_bottom_lip_x	7016
29	mouth_center_bottom_lip_y	7016
30	Image	7049

This dataset has missing values so I will train my base models on 20% of the dataset and the rest I will use afterwards.

I have explained this process in detail in the data preprocessing section.

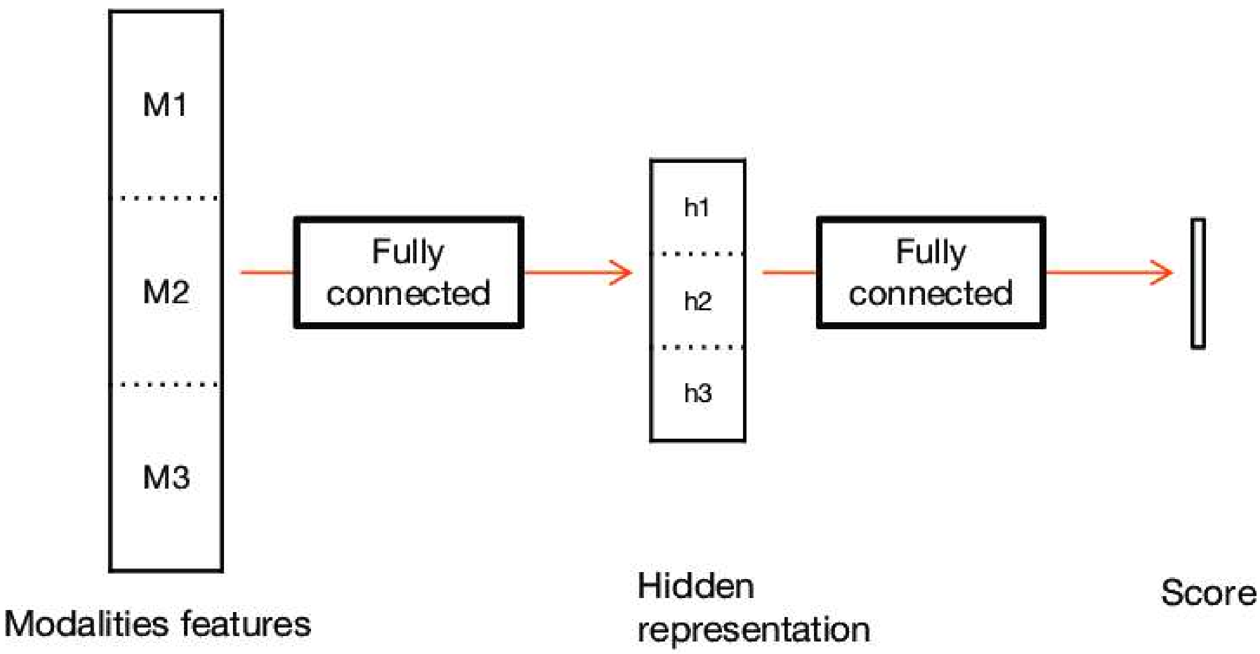
Algorithms and Techniques

For this problem, I will basically use 3 different techniques:

Fully connected model:

In this, I will use a Sequential model with 3 different layers followed by an activation function “**Relu**”, and I will also add a dropout after the first layer. I have used SGD optimizer for this using 50 epochs and a batch size of 128.

A fully connected model looks like this:



A Simple CNN:

In this, I will use a Sequential model with 3 different Conv2D layers each having a max pooling layer having a pool size and stride of 2*2. Each layer I have also added batch normalization and dropouts to avoid overfitting. At the end, I have also added 3 fully connected layers with dropouts.

In this, I have used adam optimizer having epochs set to 200 and a batch size of 128.

A CNN model looks like this:

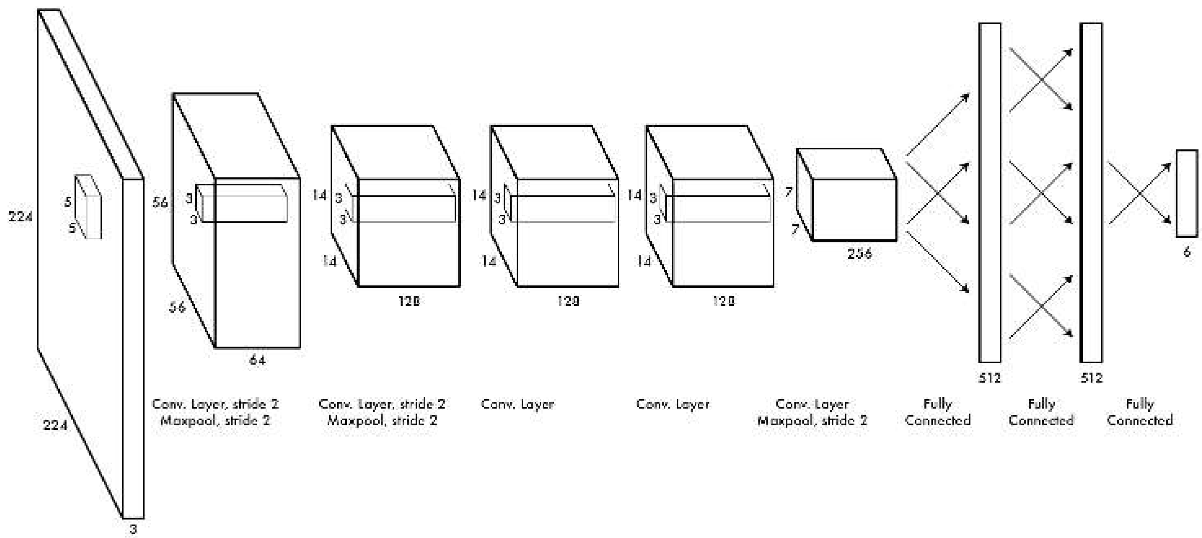


Image Augmentation:

In this, I have used the Flipping Technique to flip the images but used the same model architecture as the previous model. The number of epochs being 200.

Benchmark

I have created 3 benchmark models for this problem:

A fully connected model:

```
%%time
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
from keras.layers import Dropout

model = Sequential()
model.add(Dense(128,input_dim=X.shape[1]))
model.add(Activation('relu'))
model.add(Dropout(0.1))
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(30))

model.summary()
```

```
sgd = SGD(lr=0.01, momentum=0.9, nesterov=True)
model.compile(loss='mean_squared_error', optimizer=sgd)

hist = model.fit(X, y, nb_epoch=50,batch_size=128, validation_split=0.2,verbose=False)
```

A Simple CNN:

```
def CNN():
    model2 = Sequential()

    model2.add(Conv2D(filters=16,kernel_size=2,padding="same",activation="relu",input_shape
=(96,96,1)))
    model2.add(Dropout(0.1))
    model2.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), border_mode="valid"))
    model2.add(BatchNormalization())

    model2.add(Conv2D(32, 5, 5,activation="relu"))
    # model.add(Activation("relu"))
    model2.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), border_mode="valid"))
    model2.add(Dropout(0.2))
    model2.add(BatchNormalization())

    model2.add(Conv2D(64, 5, 5,activation="relu"))
    # model.add(Activation("relu"))
    model2.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), border_mode="valid"))
    model2.add(BatchNormalization())

    model2.add(Conv2D(128, 3, 3,activation="relu"))
    # model.add(Activation("relu"))
    model2.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), border_mode="valid"))
    model2.add(Dropout(0.4))
    model2.add(BatchNormalization())
```

```
    model2.add(Flatten())

    model2.add(Dense(500, activation="relu"))
    model2.add(Dropout(0.1))

    model2.add(Dense(128, activation="relu"))
    model2.add(Dropout(0.1))

    model2.add(Dense(30))


    model2.summary()
    model2.compile(optimizer='adam',
              loss='mse',
              metrics=['mae','accuracy'])
    return(model2)
```

```
model2 = CNN()
hist2 = model2.fit(X, y, nb_epoch=500,batch_size=128, validation_split=0.2,verbose=False)
```

A Simple CNN after Flipping the Images:

```
model3 = CNN()
flipgen = FlippedImageDataGenerator()
hist3 = model3.fit_generator(flipgen.flow(X_train, y_train),
                             samples_per_epoch=X_train.shape[0],
                             nb_epoch=300,
                             validation_data=(X_val, y_val))
```

Methodology

Data Preprocessing

By looking at this dataset we can see that there are many missing values.

```
                        index     0
0             left_eye_center_x  7039
1             left_eye_center_y  7039
2            right_eye_center_x  7036
3            right_eye_center_y  7036
4        left_eye_inner_corner_x  2271
5        left_eye_inner_corner_y  2271
6        left_eye_outer_corner_x  2267
7        left_eye_outer_corner_y  2267
8       right_eye_inner_corner_x  2268
9       right_eye_inner_corner_y  2268
10      right_eye_outer_corner_x  2268
11      right_eye_outer_corner_y  2268
12      left_eyebrow_inner_end_x  2270
13      left_eyebrow_inner_end_y  2270
14      left_eyebrow_outer_end_x  2225
15      left_eyebrow_outer_end_y  2225
16     right_eyebrow_inner_end_x  2270
17     right_eyebrow_inner_end_y  2270
18     right_eyebrow_outer_end_x  2236
19     right_eyebrow_outer_end_y  2236
20                   nose_tip_x  7049
21                   nose_tip_y  7049
22          mouth_left_corner_x  2269
23          mouth_left_corner_y  2269
24         mouth_right_corner_x  2270
25         mouth_right_corner_y  2270
26       mouth_center_top_lip_x  2275
27       mouth_center_top_lip_y  2275
28    mouth_center_bottom_lip_x  7016
29    mouth_center_bottom_lip_y  7016
30                        Image  7049
```

So I used only 20% of the dataset which doesn't have any missing values.

I will use this 20% to create my base model and after that use the entire dataset for training.

```
dtype: int64
X.shape == (2140, 9216); X.min == 0.000; X.max == 1.000
y.shape == (2140, 30); y.min == -0.920; y.max == 0.996
```

From the above figure I can see that there are 2140 samples and for X: 9216 columns [96*96] and for y: 30 columns [x and y co-ordinates each having 15 keypoints].

```
df = df.dropna()  # drop all rows that have missing values in them
```

After this we are left with only 2140 samples.

After creating my base models, I divided the 15 Landmarks into 6 different groups as shown below:

```
SPECIALIST_SETTINGS = [
    dict(
        columns=(
            'left_eye_center_x', 'left_eye_center_y',
            'right_eye_center_x', 'right_eye_center_y',
            ),
        flip_indices=((0, 2), (1, 3)),
        ),

    dict(
        columns=(
            'nose_tip_x', 'nose_tip_y',
            ),
        flip_indices=(),
        ),

    dict(
        columns=(
            'mouth_left_corner_x', 'mouth_left_corner_y',
            'mouth_right_corner_x', 'mouth_right_corner_y',
            'mouth_center_top_lip_x', 'mouth_center_top_lip_y',
            ),
        flip_indices=((0, 2), (1, 3)),
        ),
```

```
    dict(
        columns=(
            'mouth_center_bottom_lip_x',
            'mouth_center_bottom_lip_y',
            ),
        flip_indices=(),
        ),

    dict(
        columns=(
            'left_eye_inner_corner_x', 'left_eye_inner_corner_y',
            'right_eye_inner_corner_x', 'right_eye_inner_corner_y',
            'left_eye_outer_corner_x', 'left_eye_outer_corner_y',
            'right_eye_outer_corner_x', 'right_eye_outer_corner_y',
            ),
        flip_indices=((0, 2), (1, 3), (4, 6), (5, 7)),
        ),

    dict(
        columns=(
            'left_eyebrow_inner_end_x', 'left_eyebrow_inner_end_y',
            'right_eyebrow_inner_end_x', 'right_eyebrow_inner_end_y',
            'left_eyebrow_outer_end_x', 'left_eyebrow_outer_end_y',
            'right_eyebrow_outer_end_x', 'right_eyebrow_outer_end_y',
            ),
        flip_indices=((0, 2), (1, 3), (4, 6), (5, 7)),
```

I will train my model on each of these 6 groups separately.

All 6 models contains the same CNN architecture but the final output layer is adjusted for different number of outputs: for example we have a model for left eye and right eye center landmark prediction. As there are x and y coordinates for both eye centers, we have 4 nodes in the output layer of this model.

Implementation

As stated earlier I have created 3 base models with 20% of the dataset.

After that as explained in the Data Preprocessing section, I created a specialist list containing the rest of the dataset and trained model2 and model3 using the same.

With model3:

```
%%time
specialists1 = fit_specialists(freeze=True,
                               print_every=10,
                               epochs=100,
                               name_transfer_model="my_model3.h5")
```

With model2:

```
%%time
specialists2 = fit_specialists(freeze=True,
                               print_every=10,
                               epochs=100,
                               name_transfer_model="my_model2.h5")
```

Refinement

I experimented with my models quite a bit.

I increased the number of epochs from 200 to 500 for my base model2 and it subsequently made my special model2 models performance better.

I also increased the number of epochs from 200 to 300 for my base model3 but it made my special model3 performance worse.

Results

Model Evaluation and Validation

The final model ie, Special model2 was derived from the base model2 which I made using 20% of the dataset.

As stated in the previous section I experimented with the base model's by increasing its epochs from 200 to 500 and the performance was slightly better than the previous one.

Note: Please check the justification section.

This model's performance after tuning it can be trusted.

Justification

The final model's result is stronger than the base model's result.

To prove my point I am attaching my submissions from Kaggle:

Overview Data Kernels Discussion Leaderboard Rules Team My Submissions Late Submission

✔ Your submission description has been saved.

Submission and Description	Private Score	Public Score	Use for Final Score
Facial Keypoint Detection Udacity (version 11/11) 4 minutes ago by aparajit chatterjee From "Facial Keypoint Detection Udacity" Script special_model3	1.99717	2.36345	☐
Facial Keypoint Detection Udacity (version 11/11) 9 minutes ago by aparajit chatterjee From "Facial Keypoint Detection Udacity" Script model3	3.49977	3.55965	☐
Facial Keypoint Detection Udacity (version 11/11) 13 minutes ago by aparajit chatterjee From "Facial Keypoint Detection Udacity" Script special_model2	1.97373	2.32953	☐
Facial Keypoint Detection Udacity (version 11/11) 15 minutes ago by aparajit chatterjee From "Facial Keypoint Detection Udacity" Script model2	2.98762	3.11029	☐
Facial Keypoint Detection Udacity (version 9/11) 4 days ago by aparajit chatterjee From "Facial Keypoint Detection Udacity" Script special_model2_b4_alt	1.98025	2.32171	☐
Facial Keypoint Detection Udacity (version 8/11) 7 days ago by aparajit chatterjee From "Facial Keypoint Detection Udacity" Script special_model3_b4_alt	2.00341	2.38985	☐
Facial Keypoint Detection Udacity (version 8/11) 7 days ago by aparajit chatterjee From "Facial Keypoint Detection Udacity" Script model3_b4_alt	3.35700	3.41085	☐

Conclusion

Free Form Visualization

These two images shows the final models performance in comparison to its base model.

"best" - The two model results agree the most.

"worst" - The two model results disagree the most.


Good:model2:pic80
Good:special:pic80
Bad:model2:pic1116
Bad:special:pic1116
Good:model2:pic20
Good:special:pic20
Bad:model2:pic1064
Bad:special:pic1064
Good:model2:pic88
Good:special:pic88
Bad:model2:pic524
Bad:special:pic524
Good:model2:pic92
Good:special:pic92
Bad:model2:pic1206
Bad:special:pic1206

Reflection

The following steps were taken to complete this process:

1. Downloaded the dataset from kaggle.
2. Scaled the pixel values.
3. Used 20% of the dataset to train the base models.
4. Created special models by dividing the dataset into 6 different groups and trained the base models separately on them.
5. Converted the files to csv and compared their scores on kaggle.

As this was the second time apart from dog breed classifier I dealt with image classification problem, I have learned a lot from this project. The most challenging part was to create the third base model I tried using the Keras image flipping inbuilt technique but to flip the keypoints I had to flip them manually.

Improvement

There are a few ways in which I can suggest some improvements:

1. Tuning/Experimenting with the number of epochs.
2. Using different optimizers.
3. Tuning/Experimenting with batch size.
4. Using different image augmentation techniques such as shifting, scaling etc.

One technique I would I like to use in particular is OpenCV but I am not too much familiar with it.