Facial Feature Extraction in Unconstrained Environments

Steven Branson

Abstract

For this project, I investigated a few different techniques that could be used for facial feature extraction, face alignment, and facial feature tracking. My goal was to find a solution to overcome limitations of existing methods in face recognition, analysis, and tracking to generalize to uncontrolled environments of unknown lighting, camera response function and facial identity, pose, and expression. I implemented a few different popular methods for learning and fitting active appearance models (AAM) and performed experiments to quantify their strengths and weaknesses. I then devised my own method for facial feature extraction and face alignment using sparse non-linear regression, with the goal of overcoming these limitations. Lastly, I describe methodologies of how this technique can be as a preprocessing step to existing face processing algorithms.

1. Introduction

Over the last 10-20 years, faces have been among the most widely studied objects in recognition and detection, and have also been one of the areas in which computer vision has achieved the most success. Despite this, most face recognition and analysis techniques make simplifying assumptions such as fixed pose, expression, lighting, and calibrated cameras. These assumptions limit their practicality in being scalable in full to real world applications.

While there has been considerable work [10] [12] [2] in creating methods to work under variable lighting, pose, expression, and identity, most of these methods focus only on varying one or two parameters at a time and assume everything else is fixed. Results are demonstrated on datasets that adhere to these assumptions. In real world images and video, lighting, pose, expression, identity, and camera types each vary across a continuum of possible values. Each parameter is important, and conjunctively they are not separable from one another. In this way, faces still suffer from the same curse of dimensionality problem as in other problems in object recognition.

My goal of this project was to work toward extending facial feature tracking and face recognition and analysis into less restrictive environments. A generalized facial alignment and preprocessing method could be used to help make the vast literature and prior work on faces more practical to real world applications. 2D shape and appearance methods are probably the most commonly used methods for this application [18] [17] [9], and this is the approach I decided to take. Other differing methods that have received a good deal of attention can be broadly grouped into two categories: those based on local feature detectors or parts, and full blown synthetic models.

Methods that model faces as a collection of parts that are initially detected separately [22] [15] do not produce as rich an output as model-based methods, and in practice tend to produce noisy results. One reason is that faces in general do not have that many features that can be detected accurately without using context or by using local interest point detectors. The edge of the face is an important source of shape information that cannot easily be captured by part-based models. Part-based methods must then rely on the eyes, mouth, and nose as the principal sources of information. Because the eyes and mouth are extremely deformable, they are difficult to detect accurately.

On the other end of the spectrum, synthetic model based methods have achieved a good deal of success. Blanz et. al [3] employ a 3D statistical face model. On a different but not entirely unrelated approach, Georghiades et. al [10] use images captured under different illuminations to synthesize a face model. These approaches achieve good results when they work. The major drawback to these types of methods is that they require data that is captured using highly controlled methods. This then makes it difficult to apply these techniques to real images, because other types of physical processes that go into synthesizing real life images must be physically modelled by hand.

I therefore took as my approach a method based on 2D shape and appearance models, which also uses a generative model but is learned from actual instances of images we are fitting. First introduced by Cootes et al. [5], the active appearance model (AAM) is a method for modeling and fitting a linear model of shape and texture. While such methods have achieved success in facial alignment and non-rigid tracking [19], it has been observed [14] that performance degrades significantly when the AAM must be trained to handle unconstrained environments and the number of required AAM model parameters gets larger.
Many enhancements to the original AAM have been subsequently explored. Attempts have been made to enhance the AAM generative model to be multimodal [6] [4] [25]; however, introduction of a multimodal model necessarily makes the fitting process more complex and susceptible to local minima, and as a result the solution is not readily tractable to AAMs. Romdhani et al. [20] extended ASMs and AAMs using kernel PCA, which can be used to effectively model non-linearities in parameter space; however, such methods have only proven to be useful in the ASM domain, where appearance models based on local patches exhibit non-linearities due to changing pose. For AAMs, this does not result in much improvement, presumably because a linear model of shape and texture tends to be a valid model for faces.

Instead, the main problem with the AAM in unconstrained environments arises due to the fitting stage. One problem with AAM fitting is that it uses a simple linear update model that is prone to getting stuck at local minima. Hou et al. [16] modified the AAM fitting procedure to operate on a subspace on the residuals of texture difference between the image and predicted model; however, their method still uses a linear update step that is subject to all the usual problems. Cristinacce et al. [8] and Saragih et al. [21] use boosting to learn a regression function for updating model parameters; however, their methods still seem to be limited to AAMs with lower number of model parameters.

For this project, I first implemented a couple popular AAM learning and fitting methods. I devised a series of tests designed to learn what specifically makes face analysis in non-restricted environments so hard, and why the AAM tends to fail under these conditions. I found that the primary problem with the AAM fitting is the way in which the gradient with respect to the model parameters is calculated, by perturbing each model parameter individually by some fixed amount. This is problematic because with respect to the error residuals, model parameters are not actually independent. At any given point in the fitting stage, the residual error arises mostly due to a few principal model parameters, causing the lower order model parameters to be pushed off in some random direction.

I then developed a modified fitting algorithm designed to overcome these problems, by training a sparse kernelized regression model that is trained on residuals when perturbing all parameters at once. The advantages of my approach over conventional AAMs is that it can continue to adapt in complexity as necessary to find the true globarl minimum, it provides a principled technique to fitting many model parameters of different levels of importance, and it can model non-linear update rules. The method is loosely based on adapting techniques used for body pose estimation [1] and hand tracking to AAM parameter estimation.

2. Active Appearance Models

The AAM is a generative model in which the appearance of an object is modeled as linear combination of shape and texture parameters:

\[ S = \mu_s + \sum_{i=1}^{N_s} u_i s_i \]  
(1)

\[ T = \mu_t + \sum_{i=1}^{N_t} v_i t_i \]  
(2)

Here \( S \) is a vector of length \( 2M \) containing \( (x, y) \) coordinates specifying \( M \) points of interest on an object, \( s_i \) is a \( 2F \) vector which is scaled by its corresponding shape parameter \( u_i \).

The image is then warped into the mean shape \( \mu_s \) and the pixels inside define the texture \( T \) of the object, which is expressible as a \( O \) dimensional vector. It is similarly parameterized by a linear model defined by \( \mu_t \) and a set of vectors and weights \( t_i \) and \( v_i \). In this way, the appearance of an object is fully parameterized by \( \theta \) as:

\[ \theta = \begin{pmatrix} U \\ V \end{pmatrix} \]

(3)

where \( U = u_1...u_{N_s} \) and \( V = v_1...v_{N_t} \).

The values \( \mu_s, u_1...u_n, s_1...s_{N_s} \) and \( \mu_t, v_1...v_n, t_1...t_{N_t} \) are learned by hand labelling each of the \( M \) interest points in a set of \( P \) training images and performing PCA on the set of all vectors \( S_{1...P} \) and \( T_{1...P} \). Here each \( u_i \) and \( s_i \) is an eigenvalue and eigenvector of the covariance matrix of vectors \( S_{1...P} \), and each \( v_i \) and \( t_i \) is an eigenvalue and eigenvector of the covariance matrix of vectors \( T_{1...P} \).

2.1. Fitting the AAM

The AAM defines a generative model

\[ I_{gen} = Warp(\theta) \]

(4)

where \( Warp() \) is a function that does an affine warp of the texture pixels \( T \) according to the shape coordinares \( S \), and \( S \) and \( T \) are computed using equations (1) and (2). Fitting the AAM is done by minimizing a squared loss function:

\[ L = E^T E \quad \text{where} \quad E = I_{gen} - I_{src} \]

(5)

In the fitting algorithm described in Cootes et. al [5], fitting involves obtaining a rough initial estimate of the AAM parameters \( U, V \), and then iteratively searching for a local minima to the loss function using gradient descent. At each step of the optimization algorithm, we are searching for the ideal update \( \Delta \theta \), such that

\[ \frac{\partial}{\partial \theta} (Warp(\theta + \Delta \theta) - I_{src})^2 = 0. \]

(6)
In the limit as $\Delta \theta \rightarrow 0$, this evaluates to

$$\Delta \theta = -R \hat{E} \text{ where } R = (\frac{\partial E}{\partial \theta}^T \frac{\partial E}{\partial \theta})^{-1} \frac{\partial E}{\partial \theta}$$

(7)

For the purpose of computational efficiency, $R$ is computed offline. Here $\frac{\partial E}{\partial \theta}$ is a vector of partial derivatives of the error $E$ with respect to each AAM model parameter $\theta_i$ in $\theta$. $\frac{\partial E}{\partial \theta}$ is estimated offline by taking the average value of $\frac{\Delta E}{\Delta \theta_i}$ over each training image when perturbing a model parameter $\theta_i$ by some amount (I used +/- .5 standard deviations).

2.2. The Inverse Compositional, "Project Out" Algorithm

I implemented a second popular AAM fitting algorithm described in Matthews and Baker [19]. This algorithm offers much faster training and fitting time (some implementation can track faces at 300fps), while giving similar fitting results and convergence properties. The main idea behind this algorithm is that the texture parameters can be “projected out” into a precomputation, such that the fitting algorithm is a function only of the shape parameters (of which there are much fewer number than texture parameters). Precomputations are done with respect to the mean texture $\mu_t$ rather than with respect to the training images, as are computations of residual pixels. A simplified overview of the algorithm (omitting additional complexity to handle translation, rotation, and scale) is described below:

**Offline:**

- Compute the $N_s X N_s$ inverse Hessian matrix $H^{-1}$ of shape parameters, where each partial derivative $\frac{\partial E}{\partial u_i}$ is computed by perturbing the corresponding shape parameter $u_i$, and $\Delta \hat{E}$ is the difference between the predicted model image and the mean texture $\mu_t$.

**Iterate Until Convergence:**

1. Compute $\hat{E} = I_{gen} - I_{src}$
2. Update the shape parameters by $\Delta U = H^{-1} \sum_{i=1}^{N_s} [\frac{\partial E}{\partial u_i}]^T \hat{E}$

2.3. AAM Results

I tested the two fitting algorithms on a the IMM face dataset, a video sequence of a face moving through different expressions and pose, and random face images on the web. The IMM face dataset, which contains 240 images total and is composed of 6 face images of 40 different subjects at different pose, lighting, and expression. I used half the images for training and half for testing, where the people in the test set were not included in any image in the training set. The video sequence had approximately 30 frames annotated, and testing was done on different but similar video sequence. The set of images from the web was composed of 130 faces obtained from the web using a search engine. To keep the results comparable to the IMM dataset, on each testing run I picked a random 10 test images and trained on the remaining 120 images, and computed the average over 10 different test runs. Since I found that the Matthews/Baker fitting algorithm had comparable performance to Coote’s fitting algorithm and was much faster to train/fit, I am presenting my results using that algorithm.

My goal was to devise a few tests that I could use to better understand the strengths and weaknesses of AAMs and the properties of faces in general, such that I could devise an improved algorithm for the purposes of facial alignment in unconstrained environments of larger intrinsic dimensionality.

2.3.1 On the Dimensionality of Face Images

My first test was to simply learn an AAM from training data and empirically determine what are the main factors that contribute to the dimensionality of face images.

One can see in figure 1 that the primary factors which explain the variance of face images are, roughly in order of strength: lighting, pose, and expression. The 1st two principal components of the combined model roughly correspond to lighting, while the head pose is also very important and begins to come in on the second or third principal component. Expression does not play nearly as big a role in explaining the variance of the image as lighting and pose, but is still very significant. A smiling expression begins to show up in the 4th or 5th principal component.

These results seem to indicate that pose, lighting, and expression can be represented concisely using a small number of linear components in the shape/texture space. This falls in line with the conclusions in [11], that the set of pose aligned face images under all possible lighting conditions is expressible using a small number of linear components. Even a single linear component is a very close approximation to rotating the pose of the face about a fixed axis, which follows from simple 3D geometry. Expressions such as smiling have a strong prior in the shape domain, i.e. the smiling motion represents a similar type of motion for all people. Since lighting, pose, and expression are expressible using a small number of linear weights and each contribute significantly to the variance of pixels in face images, one would expect that it is feasible to extract these parameters from face images. Indeed, this is probably why the AAM fitting tends to work well with respect to varying pose, lighting, and expression.

Facial identity, on the other hand, is not as easily separa-
Figure 1. Visualization of the first 5 shape, texture, and combined principal components. The 1st row shows the mean shape and annotated feature points learned from the IMM dataset. The second row shows the first 5 shape principal components perturbed by one standard deviation from the mean. The next 2 rows show the 1st 5 texture and 1st 5 combined principal components perturbed one standard deviation from the mean.

Each intrinsic identity parameter contributes only a small fraction to the variance of face images, but there are a lot of them and collectively they matter. It is difficult to say whether a linear model of shape and texture is good or bad in representing identity. The space is intrinsically high dimensional—people simply have very subtle differences in the way that they look. The fact that identity represents a high dimensional space and is an obstacle for AAM fitting is explored in [14].

To get an idea for the intrinsic dimensionality of the different datasets, I plotted the number of principal components needed to represent a percentage of the total variance, see figure 2. We can get an intuition here as to why fitting performance is good on person-specific models such as my video sequence—it is of low intrinsic dimensionality. We can also get an idea as to why it might be worse for images on the web: these images require a greater number of shape parameters.

2.3.2 Evaluating Fitting Performance

My next test was to examine the fitting performance on each of the three datasets. As an error metric, I used the sum squared error (SSE) between the location of the feature points predicted by the AAM and the hand labelled ground truth in the test set. I set the number of parameters included in the model such that the model could explain 95% of the variance in the training set, as in [9]. The tests are done by initializing a mean face that is randomly translated, rotated,
Figure 2. Number of AAM parameters used to explain the variance in face images. The shape parameters and combined appearance parameters are shown on the left and right respectively. Plots show the number of principal components needed to model a percentage of the total variance.

and scaled and running the fitting algorithm. The results of the experiments are shown in figure 3.

We see a clear ordering of fitting performance, the video sequence has the best convergence properties, followed by the IMM dataset, followed by images on the web. In figure 4, I show typical examples of fitting performance on each of the 3 datasets.

I found that the selection of the number of AAM parameters used by the model has a big role on the convergence properties of the AAM. In figure 5, we can clearly see that when the AAM fails, it is because the fitting algorithm gets stuck on a local minimum and fails to find the true global minimum. We see that as the number of AAM parameters grows in size, the fitting algorithm doesn’t come close to being able to find the true global minimum. Although one would typically like to select a number of AAM parameters as necessary to explain, say 95% of the variance, this is not usually a good idea. Below is a list of what I determined to be the major weaknesses of AAM fitting:

1. The fitting algorithm is the major limiting factor to the AAM’s ability to generalize to environments of greater dimensionality. This occurs because as more model parameters are required by the model, the search space becomes increasingly non-linear and non-convex. Intuitively, this arises because as the number of model parameters increases, there becomes a confusion/conflict as to which combination of model parameters to update in order to explain the current model error.

2. In particular, the shape parameters are the primary culprit to the nonconvexity of the search space. I draw this conclusion because I found that if I increased the number of shape parameters above \( \approx 5 \), convergence rate started to decay, regardless of the number of shape parameters needed to explain the variance in the training set. The intuitive reason is that the AAM update rule during each iteration of the fitting algorithm is computed using the texture sampled under the current shape parameters, and as a result, the convergence results are highly sensitive to errors in the shape domain.

3. Variation in the background behind the face image is one of the reasons why the AAM fitting might not work as well on the web images as in the IMM dataset. For example, a background that is black versus a background that is white will yield opposing update rules to the gradient when precomputing the error Hessian matrix.

4. Other troublesome sources of dimensionality include varying camera response functions and lens distortion, and greater range of variation in the identity, pose, and expression.

5. Existing AAM fitting methods involve precomputing a Hessian matrix that is of fixed size equal to the number of model parameters, and in this way assume that all faces share the same linear update rules, regardless of the current model parameters.

6. The update rule is determined by measuring the residual error when perturbing a single model parameter at a time; however, when fitting an image many different parameters will contribute to the residual error simultaneously. Most of the variance in the residual error
pixels will be explained by the parameters of highest eigenvalue, and this will likely cause the parameters of lower eigenvalues to update in some random direction.

These observations suggest the need of a learning and fitting model of greater complexity. While most current methods have avoided this due to the goal of obtaining a real-time face tracker, I argue that this is unnecessary if separate appearance and motion models are learned. In the next section, I describe a different learning and fitting procedure that is motivated by these observations.

3. Fitting an AAM Using A Sparse Non-Linear Regression Model

The experiments and observations described in the previous section led me to believe that the main reason why AAM fitting fails on spaces of larger intrinsic dimensionality is the assumption of a linear update rule that can be
approximated by perturbing each model parameter independently. The parameters of lower eigenvalue, which individually do not explain much of the variance in the image but collectively are significant, get pushed off in some random direction as a result of residual error caused by more significant parameters. In addition, because the update rule is determined fully by a fixed size Hessian matrix, the AAM learning model does not continue to improve with increasing amount of training data.

This motivated a different kind of learning algorithm in which the complexity of the update model could grow however large it needs to grow in order to allow convergence to the true global minimum. A sparse kernelized regression method is appropriate for these purposes, because it can increase in complexity only if it is necessary to obtain a certain level of accuracy on the training data. Such methods have been used successfully for full body pose estimation [1] and hand tracking [23].

To train the regression model, I follow a similar procedure to the one devised by Hou et. al [16], by training on the residual error when perturbing the AAM shape parameters by a certain amount. One critical difference is that rather than perturbing just a single model parameter at a time, I perturb all parameters simultaneously according to the learned AAM model, which is a generative model defining a probability distribution over face images. Thus for each training instance, the input/output to the learner is a pair \([E_p(\Delta \theta), -\Delta \theta]\), where \(\Delta \theta\) is a vector of length \(N_s\) defining the amount each shape parameter was perturbed, and \(E_p(\Delta \theta)\) is the residual error when training image \(p\) is perturbed by \(\Delta \theta\). To normalize the residual error image, I project out the texture parameters, and thus compute \((\Delta E_p(\Delta \theta))\) as:

\[
(\Delta E_p(\Delta \theta) = (I_p - (T(I_p - \mu_t)T^{-1} - \mu_t))
\]  

(8)

Where \(I_p\) is the result of sampling the texture of \(p\), an image when perturbing by \(\Delta \theta\). \(T\) is the matrix of texture parameter eigenvectors, and \(T^{-1}\) is the pseudo-inverse of \(T\).

Because it would be difficult to learn a regression model that can go in one step from any arbitrarily distant location away from the true face position, I break the process into multiple stages which progressively improve the fitting accuracy. In the first stage, I assume that I may be very far from the correct solution, say within \(\sigma_1 = 4\) standard deviations. I train a regression model \(M_0\) on pairs \([E_p(\Delta \theta_1), -\Delta \theta_1]\), where each \(\Delta \theta_p\) is drawn i.i.d. from the distribution \(p(\Delta \theta_1|\sigma_1)\). I then train a sparse regression model that is designed to take me to within \(\sigma_2\) of the true solution, where \(\sigma_2 < \sigma_1\). I repeat the process to learn \(N\) different regression functions, where \(\sigma_N\) is below some acceptable threshold. The separation into multiple stages of regression allows for better fitting accuracy and sparser regression models.

When I implemented this algorithm, I found that at each stage of the learning method, only some model parameters contribute to the residual error in a significant enough way that an accurate regression function can be learned. I therefore slightly modified this approach to only regress upon the model parameters that contribute to at least \(\epsilon\) percent of the residual error in the training images. The model parameters that don’t meet this criterion will be updated in a later regression stage once they do contribute to a high enough percentage of the residual error of the variance.

Pseudo-code of the learning and fitting algorithm is shown below:
AAMLearnRegression(D, Q, σ1, σN)
\( s_{1...N_t} \leftarrow σ_1, t \leftarrow 0 \)
WHILE \( s_{1...N_t} > σ_N \)
FOR \( t = 1 \) to \( Q \)
\[ E(Δθ_t), -Δθ_t] \leftarrow \text{RandomResidual}(D, s_{1...N_t}) \]
\( O^t \leftarrow \text{Outputs With Enough Variance} \)
\[ [W^t, V^t] = \text{LearnRVM}([E(Δθ_t)], -Δθ_t, 2s_{O^t}), t \leftarrow t + 1 \]
RETURN \( [W, V, O, t] \)

Here \( D \) is the dataset of annotated training images, \( Q \) is the number of random samples (input data and targets) to be fed to train the regression function, \( O^t \) is a list of AAM model parameter indices that will be updated at this regression learning stage, and .2 and .5 are tweakable parameters that effectively govern the sparsity of the regression model and the effects of overfitting. \( \text{LearnRVM}(\cdot) \) takes as parameters the input data and their corresponding targets and a prior on the target variance of each parameter, and returns a sparse set of basis vectors \( V^t \) and their corresponding weights \( W^t \), where each \( V^t \) is an instance of \( \theta_{1...Q} \). A face image can then be fit using the method outlined below:

AAMFitRegression(I, W, V, O, t)
Initialize \( \theta \) from a face detector
FOR \( t = 1 \) to \( Q \)
\( I_p \leftarrow \text{Sample texture in image I under } \theta \)
\( E \leftarrow (I_p - (T(I_p - μ_t)T^{-1} - μ_t)) \)
\( Δθ_{O_t} = ∑_j W_j^t K(V_j^t, E) \)
\( θ_{O_t} \leftarrow θ_{O_t} + Δθ_{O_t} \)
RETURN \( \theta \)

Here \( K(x_i, x_j) \) is a kernel function.

### 3.1. Sparse Kernel Machines

While any off the shelf regression method could be used, I decided to use a Relevance Vector Machine (RVM) [24] using a Gaussian kernel. I won’t argue as to whether or not this most appropriate regression technique or kernel for this application. The choice of an RVM was primarily motivated by the availability of source code [23] with multivariate output capability. In addition, I simply used raw residual pixels as the input data to the regression function. A feature space mapping or dimensionality reduction technique would probably yield a result that is more accurate and faster to compute. A brief overview of the standard RVM regression algorithm is below. For a more detailed description, see Tipping [24].

Like a support vector machine, the RVM seeks a predictor model
\[ y(x) = \sum_{i=1}^M w_i φ_i(x) \] (9)
that is a linear combination of nonlinear basis functions \( φ_i(x) \). The basis functions are given by kernels
\[ y(x) = \sum_{n=1}^N w_n K(x, x_n) + b \] (10)
where \( x_n \) is a training example, \( b \) is a bias term and \( K(x, x_n) \) is a kernel function, in my case \( e^{-γ|x-x_n|^2} \).

It seeks an MLE solution to the weights \( w \) and hyperparameters \( α \) for the likelihood function
\[ (p(t|X, w, σ^2) = \prod_{n=1}^N p(t_n|x_n, w, σ^2) \] (11)
where
\[ (p(w|α) = \prod_{i=1}^M N(w_i|0, α_i) \] (12)
where \( t \) is an \( N \) dimensional vector of target outputs and \( X \) is a matrix of \( N \) training examples. The solution is found by an iterative solution alternating between finding the MLE solution to \( p(w|t, X, α, σ^2) = N(w|m, Σ) \), which yields
\[ m = σ^{-2} ΣΦ^T t \] (13)
\[ Σ = (\text{diag}(α) + σ^{-2} Φ^T Φ)^{-1} \] (14)
and to \( p(t|X, α, σ^2) = \int p(t|X, w, σ^2)p(w|α)dw \), which yields
\[ \frac{1 - α_i Σ_{ii}}{m_i^2} \] (15)
The defining feature of the RVM is that it provides a separate hyperparameter for each weight. It can be shown that a proportion of the hyperparameters \( α \) will be driven to infinity, and the corresponding basis functions can be pruned from the model. This leads to the sparsity of the learned model.

### 3.2. Comparisons to Traditional AAM

The ways in which my method differs from the traditional AAM fitting methods are as follows:

1. As in [16], it predicts an update of the shape parameters directly from the residual pixels using regression.
2. It avoids the assumption that each parameter affects the model error in a way that is independent from the current model state, because I train on instances where all model parameters are simultaneously perturbed rather than a single parameter at a time.
3. It provides a principled way to avoid updating parameters for which there is currently not enough information in the residual error to predict a valid update.

4. Unlike other methods which use an update rule of fixed size, it can scale in complexity as necessary to explain the training data and therefore continues to improve with more training data.

5. By using the kernel trick, it can effectively model non-linearities in pixel space.

6. It is a non-iterative approach and runs in a fixed number of regression stages.

7. It can be trained to learn a mapping which respects the probability density of our model parameters by drawing training examples drawn from this probability distribution.

The main drawback to my approach is the computational cost for both training and fitting.

3.3. Results

I was able to get my algorithm to work on small datasets of around 20 images. Since the training time can take very long, I still need to tweak a few parameters and then train on a larger dataset. I am hoping to have actual results soon.

4. Applications

The ability to robustly fit an AAM in unconstrained environments can provide a number of useful applications for the purposes of face alignment and realtime tracking.

4.1. Converting Face Images to Canonical Form

Many existing solutions in facial recognition, expression recognition, etc. make assumptions such as fixed pose, expression, lighting, calibrated cameras, or prior registration of all relevant people in a database. If any of these assumptions does not hold, the methods drastically decay in performance or fail entirely. The ability to provide robust facial alignment would help make these solutions more practical.

An AAM can be used to convert a face image to a canonical form, where any desired combination of pose, expression, lighting, or identity can be removed by projecting them out of the model. This is achieved by first solving for the AAM parameters using the sparse regression model described in this writeup, to yield $\theta$. The training set $D$ can be partitioned into two sets $D_c$ and $D_u$, where images in $D_c$ are in canonical form and $D_u$ are not. A set of AAM shape and texture parameters $\theta_e$ can be learned by doing PCA on just the images in $D_c$. Those parameters can be projected out from the images in $D$, and a subsequent run of PCA will yield $\theta_u$. The canonical image can be computed by projecting $\theta$ into the space spanned by $[\theta_e, \theta_u]$. The parameters $\theta_u$ can then be projected out of the source image to result in an image in canonical form.

If images in the training set are annotated by pose/expression/lighting/identity, by a similar procedure, the AAM parameters can be partitioned into separate parameters for pose, expression, lighting, and identity, and semantically meaningful values for those attributes can also be extracted.

4.2. Building a Robust Facial Feature Tracker

Existing AAM-based facial feature trackers are able to run in realtime, but tend to only work well if the model is trained specifically on the person being tracked. Although my proposed algorithm is much slower than traditional AAM fitting algorithms, it can still be used for the purposes of realtime tracking. This can be done by creating an annotated training set of video sequences of different people. A specialized AAM can be trained on just the change in appearance between labelled frames in a video sequence. This can define a probabilistic motion model. This motion model can be low dimensional, because identity—the primary source of dimensionality—has been removed from the model. The model can be fit using the sparse regression method, and can then be added to the motion model (replacing the mean shape and texture). Existing fast tracking methods [19] can then be used for tracking.

5. Conclusion

For this project, my goal was to work toward creating a facial alignment and facial feature tracking algorithm that is robust to pose, lighting, expression, identity, and camera parameters. I decided that a 2D shape and appearance model was the most appropriate technique for this purpose. I implemented a couple of different traditional methods for AAM learning and fitting, and ran experiments to analyze their weaknesses. I then created a new algorithm designed to overcome these weaknesses. Finally, I described a couple of different applications where this method would be of practical value.

References


