Learning Three-Dimensional Embeddings on Fixed-Topology Face Meshes

Bradley Qu
Farhan Toddywala
Rahul Malayappan

Abstract

We explore learning a correspondence between facial images and vertex positions on a fixed-topology face mesh. By vectorizing the representation of the face mesh, we can run and compare Ridge Regression, KNN, and ResNet in their respective abilities to regress on a face image to face mesh correspondence.

1. Introduction

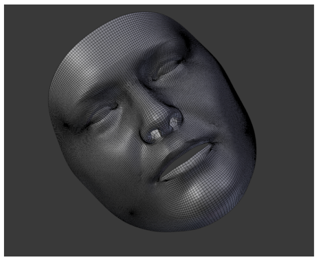
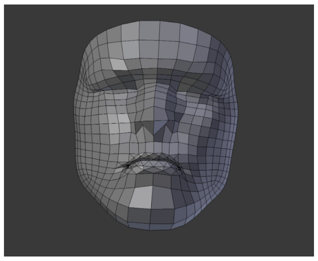
The problem of 3D facial reconstruction is not new[3], and various methods involving mesh generation have been proposed. Often, we want to generate meshes using image data; this seems especially suited to the problem of facial reconstruction, since faces follow a broad general structure, and it is plausible that we can extract features of a face from image data alone.

Often, 3D meshes generated from scans or other reconstruction methods may suffer from ”bad” topology. Topology is the connectivity structure of the edges and vertices in the mesh, and it is desirable for a mesh's topology to have regularity in face size and edge direction. ”Bad” topology is often haphazard, and may make it difficult to perform artistic operations such as texturing, detail enhancement through sculpting, or animation. Thus, artists may have to go through a tedious retopology process in order to convert a scan into a usable mesh. This suggests that it is an interesting problem to consider generating a mesh with desirable topology from the outset; one way to do this is by predetermining a set topology, and specifying a mesh by specifying the individual vertex positions on the fixed topology. In the case of face models, this makes sense as human faces follow similar structure, and can be well described by a fixed topology.

There is another way in which we can view face reconstruction in this manner; since the topology is fixed, we can imagine that it corresponds to some already-existing model, and we could imageine that a reconstruction method onto a fixed topology could allow for video-controlled animation and posing of a face.

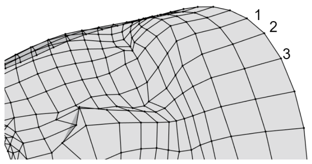
2. The Dataset

For our dataset, we used the Basel Face Model[1], a ”probabilistic morphable model” with various tunable parameters that outputs a face mesh with a fixed predetermined topology. By randomizing the various parameters, we were able to generate 5000 faces with random shape, expression and color. In addition to the 5000 full-resolution faces with about 28000 vertices, we decided to create a downsampled dataset of lower-resolution faces for the purpose of performance. To maintain regular topology, we avoided using decimation practices such as quadric error-based methods, and decimated by using inverted Catmull-Clark subdivision. As the Basel Face Model's edge structure is technically triangular, we had to manually retopologize a reference to quad topology for reference; we used this topology template for each mesh and then applied the decimation, and this resulted in a dataset where each face had about 564 vertices. The two resolutions are visualized below:

For each face, we additionally generated 128 by 128 renders for training. The colors on the Basel face model are per-vertex, and so we set up materials with vertex coloring. Originally, we rendered with simple shading and basic randomized directional lighting on a black background, with fixed ambient lighting. After the original benchmarks suffered because of this, we upgraded to more realistic skin shading using subsurface scattering and randomized specular[2], in addition to adding background Perlin noise, textures for the inside of the mouth, and more advanced randomized directional and environment lighting. The mouth is randomized between open and closed, and is just generated from a simple image texture. One picture from each dataset is shown below:

To generate the mesh dataset to be fed into the regression tasks, we encoded the meshes into data vectors. This is simple on a fixed topology mesh, since there exists a consistent ordering to the vertices:



For example, we can always correspond the top left vertex to vertex 1, and the vertex to the right of that one to vertex 2, and so on, regardless of the actual positions of the vertices. If this labeling of vertices gives vertex positions $\mathbf{v}_1, ..., \mathbf{v}_n$, we can stack the vertex positions into a data vector

$$\mathbf{d} = \begin{bmatrix} v_{1x} & v_{1y} & v_{1z}... \end{bmatrix}^\top$$

This ease of translation into a data vector is another advantage of using the fixed-topology meshes, and turns our task of facial reconstruction into a regression task from $\mathbb{R}^{128\times128}$ to $\mathbb{R}^{3n}$ on n vertices. On more general topology meshes, the discrete degrees of freedom with respect to the connectivity structure make it difficult to map meshes to data vectors.

3. Benchmarks

In order to better understand the data and to provide benchmarks for the Convolutional Neural Network approaches, we have experimented with more traditional Machine Learning models. Primarily, we focused on Ridge Regression and K-Nearest Neighbors Regression. For each of these models, we performed Cross-Validation on between 3 and 5 folds for better hyperparameter tuning. All losses/errors graphed and described are the average Mean Squared Error over a set of inputs to a trained model.

3.1. Dataset Formatting

For the traditional Machine Learning models, all spatial information is lost in the inputs as they are flattened out. For the black background dataset, benchmarks are performed on grayscale images since there is no color in the training data. These vectors flattened out from $\mathbb{R}^{128\times128}$ to $\mathbb{R}^{16384}$. The varied background dataset input vectors flatten out from $\mathbb{R}^{128\times128\times3}$ to $\mathbb{R}^{49152}$. We append a bias term of 1 to each input vector for a final input vector of $\mathbb{R}^{16385}$ and $\mathbb{R}^{49153}$ for each dataset respectively. The formatting of the output vector, given a target of n vertices, is a vector in $\mathbb{R}^{3n}$, the same as for the CNN. All of the following benchmarks are performed on the 564 vertex targets. Benchmarks on the 28588 vertex targets are in the section on Dimensionality Reduction.

3.2. K-Nearest Neighbors Regression

In KNN regression, the k nearest input vectors in the training set to a given input are found, and the weighted We chose weights inversely proportional to the Euclidean distance from the input. In the case where the Euclidean distance is 0, the weight is set to 1 and all other weights are set to 0. We have provided 2 graphs below of the Cross-Validated performance of KNN Regression, one on the preliminary dataset with a black background and one on the final dataset with varied backgrounds.

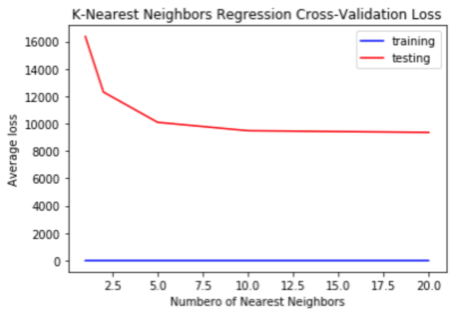


Figure 1: K-Nearest Neighbors Regression on 564 Vertex Target. Trained on Black Background dataset

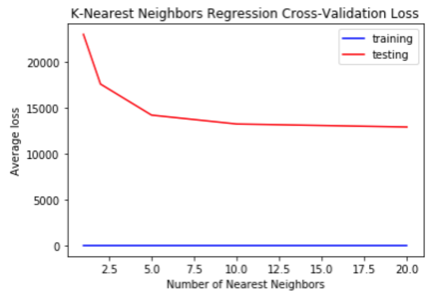


Figure 2: K-Nearest Neighbors Regression on 564 Vertex Target. Trained on Varied Background dataset

Obviously the training error is always 0 since the testing point exists in the training set. The loss flattens out to around 9000 once we check 5 or more of the closest neighbors with the black background dataset. The loss flattens out to around 12500 with the varied background dataset after the same number of neighbors. This model took the longest to train of any of the benchmarks. Even with the KDTree and BallTree optimizations, K-Nearest Neighbors processing speed was much slower than any linear technique due to the large input vectors.

3.3. Ridge Regression

Using Ridge Regression was necessary for this dataset since we had only 5000 faces but 49153 features. The complexity of Least Squares directly, where n is the number of samples and d is the number of features is $O(d^3 + d^2n)$ (for matrix inversion and multiplication). The complexity of Kernel Least Squares (using Ridge Regression) is $O(n^3 + n^2d)$. Of course, not all solvers will directly perform matrix inversion, but this analysis tells us that there are significant performance increases for using Kernel Ridge Regression (with a linear kernel), since it scales with the number of samples rather than the number of features. And for us $d >> n$.
Of course there is the matter of the ridge penalty. To optimize for this, we performed cross validation with 5 folds on various α values (where α is the regularization parameter). We also performed tests on the data by regressing on it directly, and regressing on it after centering. Here are those results.

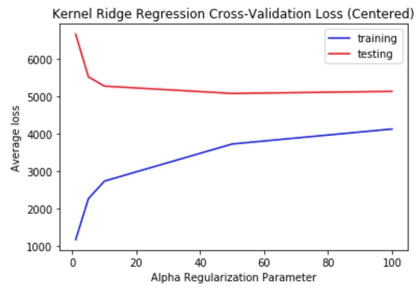


Figure 3: Centered Kernel Ridge Regression on 564 Vertex Target. Trained on Black Background dataset

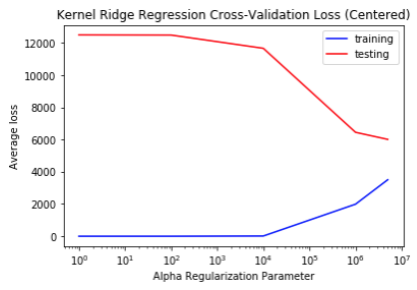


Figure 4: Centered Kernel Ridge Regression on 564 Vertex Target. Trained on Varied Background dataset

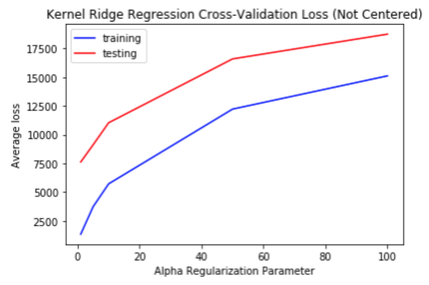


Figure 5: Unentered Kernel Ridge Regression on 564 Vertex Target. Trained on Black Background dataset

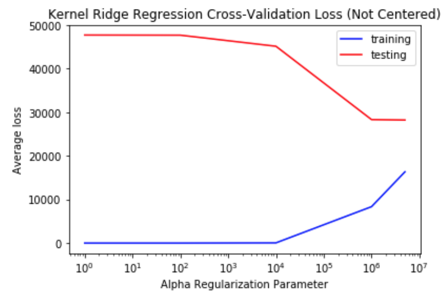
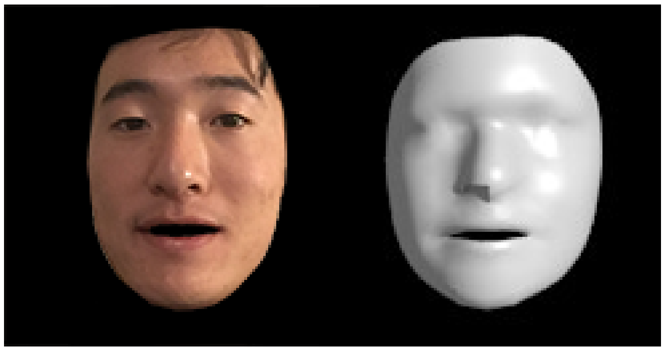


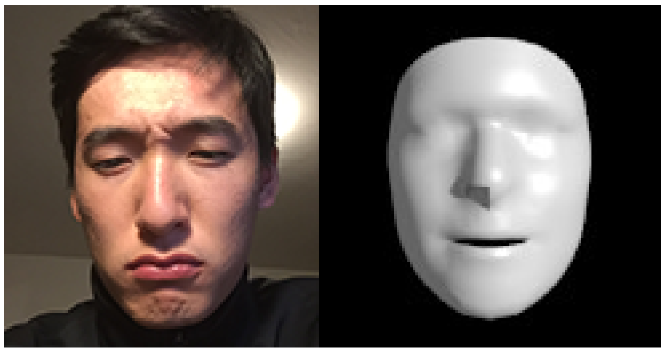
Figure 6: Uncentered Kernel Ridge Regression on 564 Vertex Target. Trained on Varied Background dataset

Clearly, centering the data was very helpful in the regression, performing with a loss of around 5000 less than the uncentered versions. The loss for the black dataset flattens out around 5000 and the loss for the varied dataset flattens out around 6000.
The varied dataset required significantly more regularization since we did not convert it to grayscale first (the black background dataset was grayscale since the colors were

only black and white). This added 3x as many features, requiring much more regularization. For the black background dataset, $\alpha = 50$ sufficied, but for the varied background dataset $\alpha = 5 \times 10^6$ was needed to combat overfitting. These values of α were used for all subsequent tests on each dataset.
All of these tests were on the 564 vertex targets. To see benchmark performance on the 28588 vertex dataset see the section on Dimensionality Reduction.

3.4. Benchmark Example Results





In both cases, Linear Regression does quite well in fitting the shape of the face in the image. Details like the forehead, nose and cheeks are fit quite well. However, it seems that in the case where there is a background, fitting the orientation of the mouth suffers. This is quite common among the other Linear Regression results: the shape of the face save for the mouth is well fit, but the mouth expressions sometimes suffer or are exaggerated.

4. Dimensionality Reduction

Detailed 3D Face Mesh models require far more than 564 vertices to be useful in practice. These meshes can use orders of magnitude more vertices. Both linear and CNN architectures possess a fully connected "final layer" to train on these mesh vertex targets, which can quickly become a bottleneck in learning if the output is too large. This is especially true if one is training without state of the art hardware. Therefore, we thought it would be interesting to explore dimensionality reduction techniques to see how we can reduce this bottleneck.

4.1. PCA

Since PCA has a trivial and easy way to reconstruct a projected vector our tests focused on using PCA. Other approaches such as CCA or tSNE did not have trivial inverse transformations we could use, so a linear, unsupervised approach like PCA would have to do. To start, we looked at the singular values for the 564 and 28588 vertex datasets.

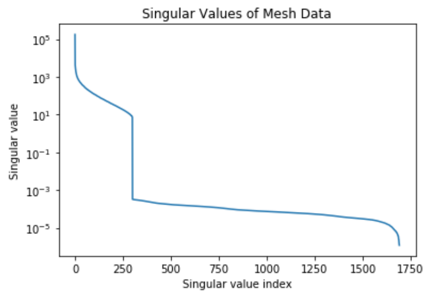


Figure 7: Singular Values of 564 vertex dataset target (5000 Meshes).

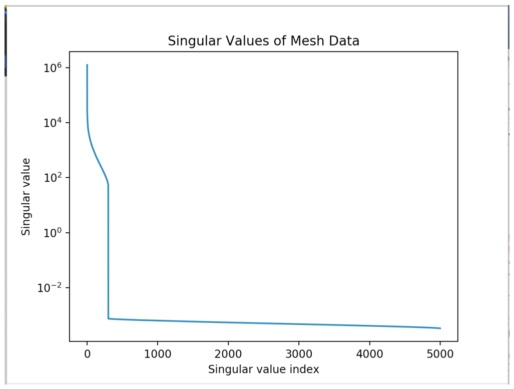


Figure 8: Singular Values of 28588 vertex dataset target (5000 Meshes).

For both datasets, the singular values drop off very quickly around 200 dimensions.

4.2. Linear Benchmark

These benchmarks were performed with $\alpha = 50$ and $\alpha = 5e6$ respectively on the black and varied background datasets

In both cases, due to the regularization and the nature of the singular values of the dataset, the loss flattens out

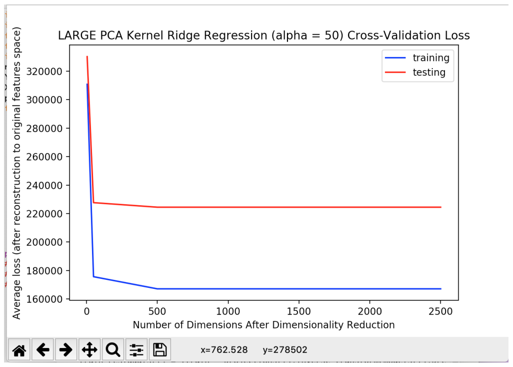


Figure 9: PCA-Ridge Regression on the 28588 vertex targets. Performed on the black background dataset.

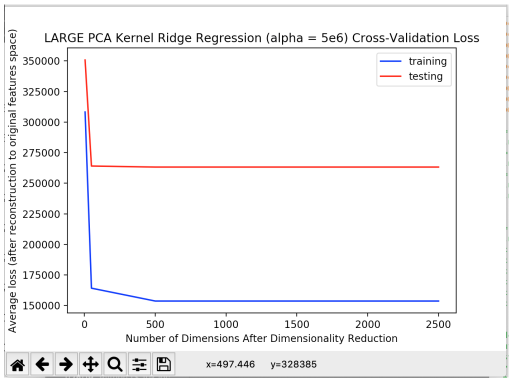
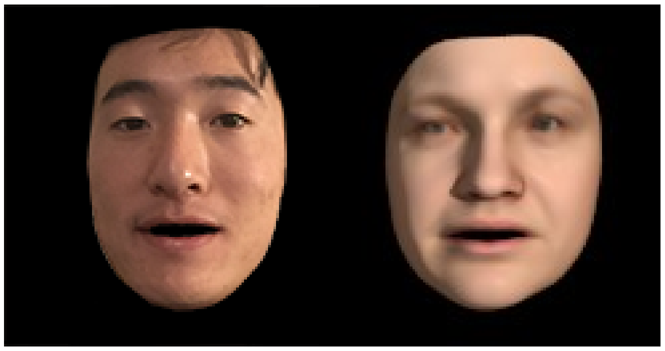
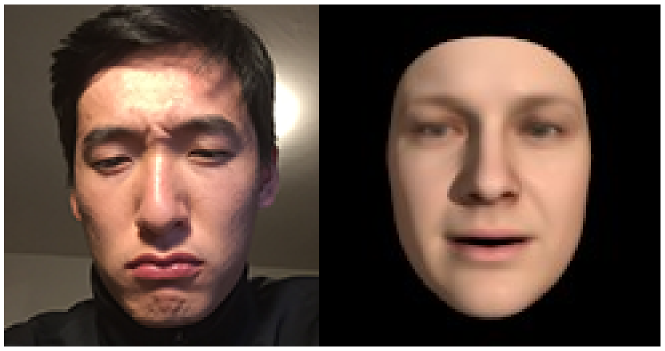


Figure 10: PCA-Ridge Regression on the 28588 vertex targets. Performed on the varied background dataset.

between 100 and 500 dimensions. The (converged) loss for the black background dataset is approximately 220000 and the (converged) loss for the varied background dataset is approximately 260000.
Note that the PCA transform was fit using the training data only.

4.3. Benchmark Example Results





Both of the above images were created by regressing on a target reduced from 85764 to 500 dimensions. Again, face shapes are fairly decent, but the mouth performance is give or take. Experiments we performed showed no discernible quality changes after 100 dimensions, indicating that we can save a large amount of processing time and space for the same quality mesh by reducing the dimensionality of the meshes before regressing on them.

4.4. Nonlinear Dimensionality Reduction

We attempted to use Kernel PCA as a nonlinear dimensionality reduction technique. We used the polynomial (degree 2), radial basis function and cosine kernels. All of them performed poorly compared to linear dimensionality reduction, with the polynomial kernel performing the best by a small margin. The lowest error achieved during tests with the 564 vertex dataset was around 7000 with 50 dimensions.

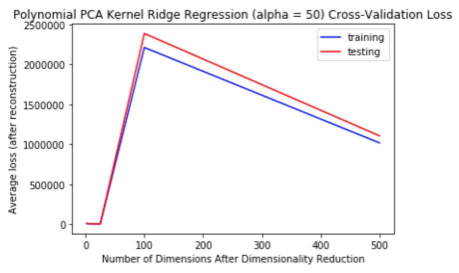


Figure 11: Polynomial (Degree 2) PCA Ridge Regression on 564 vertex target

Similarly, using these Kernels for the regression itself did not work well either. The lowest loss achieved with the RBF Kernel was around 10000.

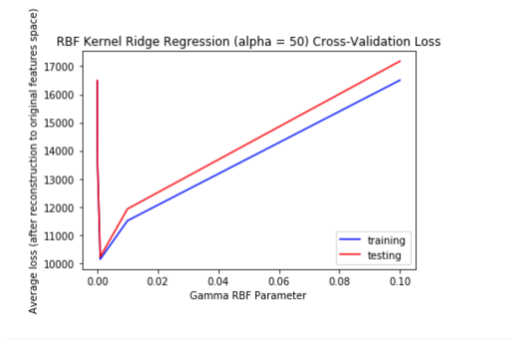


Figure 12: RBF Kernel Ridge Regression on 564 vertex target

It seems as though for this problem, a linear structure works best without any extraneous prior information about the images or the meshes.

5. Convolutional Neural Network

Even with PCA, Ridge Regression and KNN become impractical once dataset dimensions and sizes get large. This is especially visible when working with datasets like ImageNet. Similarly, the intended applications of facial reconstruction require high resolution inputs and outputs. Thus, CNNs which are able to learn complex functions with significantly fewer parameters in the long run were used.

5.1. ResNet

Since the problem can be simplified to a one input one output regression task, special neural network architectures were not required. ResNet was selected for its advantages over conventional CNNs without significant added complexity. ResNeXt was also considered, but it did not show a significant performance difference over standard ResNet.

Before training the model, the data is first pre-processed. Each mesh is represented by a vector $\mathbf{m} \in \mathbb{R}^{3n}$ vector where n is the vertex count. Each vertex element in $\mathbf{m}$ is centered and normalized by vertex mean and mesh variance respectively. This enforces the network to learn the deltas between faces rather than absolute positions of mesh vertices.

The ResNet variants tested are 18, 34, 50, and 101. Since ResNet is generally used for classification tasks, some minor modifications were made. Input is an RGB image ($128 \times 128 \times 3$). Data augmentation includes saturation (± 0.5), contrast (± 0.5), brightness (± 0.5), hue (± 0.05), translation (± 0.06), and rotation (± 4). The neural network is trained to convergence ($\approx$ 100 epochs) using standard SGD with momentum and no regularization. The output layer is expanded to fit a flattened face mesh (3×564 for small output and 3×28588 for large). The loss function is changed to MSE loss:

$$l = \frac{||\mathbf{y} - \mathbf{o}||_2^2}{3n}$$

where $\mathbf{y}$ is the centered and normalized target mesh vector, $\mathbf{o}$ is the model output, and n is the vertex count.

To interpret the output, the variance must be re-multiplied and the mean re-added. These were calculated in the pre-processing step although the experimental setup still functions using a global mean and variance given that the provided dataset is a representative subset of the global population.

5.2. Resnet PCA

Training the ResNet directly on a flattened mesh has the problem of an overly high dimensional solution space that makes finding global minima difficult. Consequently, another ResNet configuration was tested with additional pre-processing of taking the a PCA of the mesh population and taking a reduced count of principal vectors ($\approx$ 75). The image label meshes are then projected onto these eigenvectors to produce $\mathbb{R}^k$ response vectors where k is the pc count used. Output of the model is reduced in size from $3n$ to k. MSE loss is now taken on the response vectors instead of directly on the mesh.

5.3. ResNet Performance Analysis

The performances of each ResNet variant (18, 34, 50, 101) were recorded and compared. The results are depicted in Figure 13, Table 1.

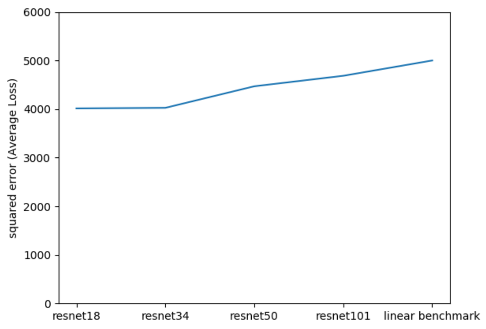


Figure 13: ResNet on 564 vertex target

ResNet Variant	Training Loss	Test Loss 564 Vertices
18 PCA(75)	3765.83	4103.27
18	3746.89	4014.25
34	3698.19	4025.36
50	2201.27	4469.1
101	2119.64	4685.12

Table 1: Average final loss ResNet variants

As opposed to the linear benchmark, ResNet performs significantly better. Increased depth, however, did not improve the testing accuracy. Instead, the deeper networks were prone to more severe overfitting. Regularization and dropout were attempted to reduce overfitting with no significant improvements. Custom narrower ResNets were tested as well which significantly reduced overfitting but also increased final test accuracy.

5.4. ResNet Final Results

The best of the tested configurations were then trained on the 28K vertex models to produce higher quality outputs. The best result was achieved with ResNet18 trained over 150 epochs with batch size 50 and a 75 principal component projection. This resulted in a loss of just under 204K per face (Ridge Regression at 220K).

5.4.1 Sample Inputs

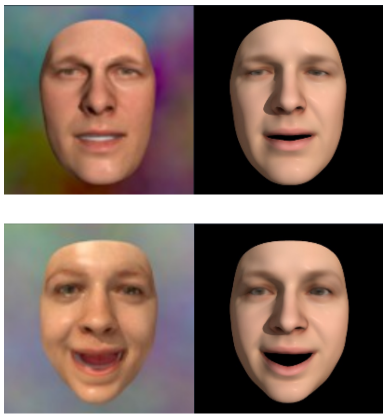


Figure 14: rendered input(left), output(right)

The model responds well to generated test faces as shown in Figure 14. Notice that the net is able to reconstruct general structure and pose very well. It, however, lacks in minor details like dimples in the lower generated sample.

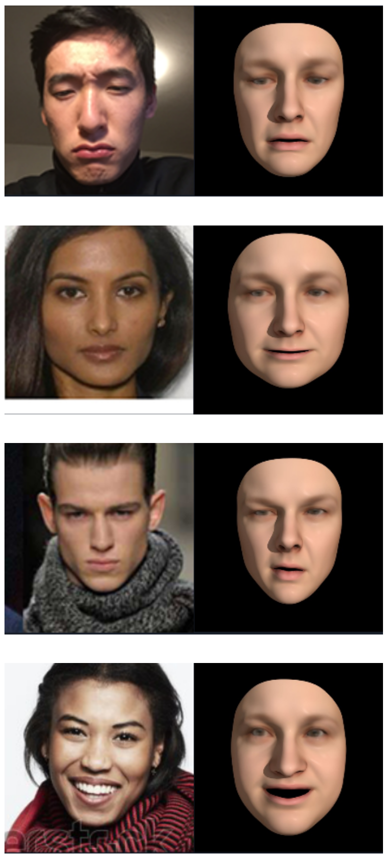


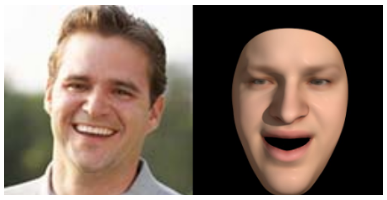
Figure 15: real input(left), output(right)

Unlike Ridge Regression, ResNet is able to generate facial meshes from real world data. This is thanks to built int data augmentation and constrained parameters.

5.4.2 Failure Cases

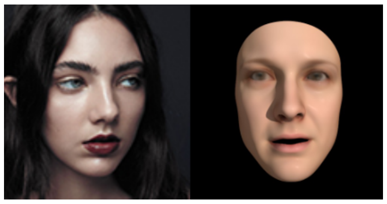
In general, we believe that the failure cases of our model are due to the lack of generality in our dataset. Specifically, our dataset had little variation in skin color, camera angle, and presence of facial hair. In addition, our face models have no neck, hair or ears, which proved troublesome when trying to regress on real photos.

One prominent failure case involved exposed necks: the model would often interpret necks as extensions of chins, causing incorrect results on the mouth and chin:



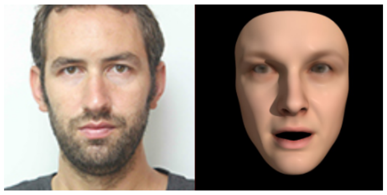
Introducing extraneous features like necks and hair are problematic given the size of the dataset; doing something like putting a flat picture of a neck or hair behind the face meshes is prone to error with respect to different faces, as the image may not line up properly. One possible improvement is to extend the Basel topology, and hand-model neck and hair meshes, upon which the models in the dataset are projected for image rendering. However, this is tedious, and requires difficulties such as hair shading and extrapolating how the rest of the mesh should be textured.

Another failure case involved camera angles; all of our camera angles were head-on, and this meant that the model couldn't adapt to images from the side:



To remedy this, we would need to render from different camera angles, but this poses problems as we would need to update the mouth rendering to work with different angles; likely we would have to manually model a mouth, and somehow fit it to an arbitrary face.

A last important failure case is that of facial hair; this is also a similar issue to the lipstick in the last picture:



This causes issues in the mouth, and it is likely simply because none of the meshes have anything resembling facial hair. This is indicative of a larger problem in our dataset, which is that the color of the meshes is quite uniform. Skin color is almost entirely Caucasian, and there is little variation in lip color and eye color. It is possible to experiment with modulating the per-vertex colors by identifying the vertices of each part of the face manually, and somehow randomizing their vertex colors. It may also be worthwhile to explore some sort of texture synthesis procedure. However, these are quite complicated, and likely highly nontrivial to do in a satisfactory manner.

6. Conclusions

In many cases, our Resnet regression performs very well on fitting facial shapes to the Basel topology. Nevertheless, there is still work to be done on improving the dataset to be more robust to different scenarios that are possible in real-life pictures. However, such improvements are not easy, and likely require significant work involving manual modeling and texturing.

References

[1] *A 3D Face Model for Pose and Illumination Invariant Face Recognition*, Genova, Italy, 2009. 1

[2] E. I. Haro A., Guenter B. Real-time, photo-realistic, physically based rendering of fine scale human skin structure. 2001. 1

[3] A. S. Jackson, A. Bulat, V. Argyriou, and G. Tzimiropoulos. Large pose 3d face reconstruction from a single image via direct volumetric CNN regression. *CoRR*, abs/1703.07834, 2017. 1