6.867 Final Project: Comparing Machine Learning Methods for Detecting Facial Expressions

Vickie Ye and Alexandr Wang

Abstract

In this project, we compared different methods for facial expression recognition using images from a Kaggle dataset released as a part of an ICML-2013 workshop on representation learning. We found that classification using features extracted manually from facial images using principal component analysis yielded on average 40% classification accuracy. Using features extracted by facial landmark detection, we received on average 52% classification accuracy. However, when we used a convolutional neural network, we received 65% classification accuracy.

1 Introduction

Detecting facial expressions is an area of research within computer vision that has been studied extensively, using many different approaches. In the past, work on facial image analysis concerned robust detection and identification of individuals [5]. More recently, work has expanded into classification of faces based on features extracted from facial data, as done in [6], and using more complex systems like convolutional neural networks, as done in [1] and [2]. In our project, we compared classification using features manually extracted from facial images against classification using a convolutional neural net, which learns the significant features of images through convolutional and pooling neural layers. The two manual feature extraction methods we explored were principal component analysis (PCA) and facial landmark detection. These extracted features were then classified using a kernel support vector machine (SVM) and a neural network.

1.1 PCA

In facial recognition, PCA is used in generating a low-dimensional representation of faces as linear combinations of the first k eigenfaces of training data. Eigenfaces are the eigenvectors of the training data's covariance matrix; the first eigenfaces are the vectors along which the training data shows the highest variance. Thus, we can express a facial image vector of thousands of pixels in terms of the linearly independent basis of the first k eigenvectors.

1.2 Facial Landmark Detection

Facial landmarks can be extracted from facial images into lower-dimensional feature vectors. The implementation that is used in this project uses the approach described in [3]. Kazemi et. al. uses localized histograms of gradient orientation in a cascade of regressors that minimize squared error, in which each regressor is learned through gradient boosting. This approach is robust to geometric and photometric transformations, and showed less than 5% error on the LFPW dataset.

The facial landmarks (eyes, eyebrows, nose, mouth) are intuitively the most expressive features in a face, and could also serve as good features for emotion classification.

1.3 Multiclass Classification

Support vector machines are widely used in classification problems, and is an optimization problem that can be solved in its dual form,

$$\max_{\alpha \in \mathbb{R}^n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y^{(i)} y^{(j)} K(x^{(i)}, x^{(j)})$$

$$\text{s.t. } 0 \leq \alpha_i \leq C$$

$$\sum_i \alpha_i y^{(i)} = 0$$

where C parameterizes the soft-margin cost function, and $K(x^{(i)}, x^{(j)})$ is our kernel function, which allows us to model nonlinear classifiers. For our multiclass classification problem, we use the nonlinear radial basis kernel function,

$$K(x, z) = \exp\left(\gamma(x - z)^2\right)$$

where γ is a free parameter. The performance of the kernalized SVM is highly dependent on these free parameters C and γ.

Neural networks are also widely used in multiclass classification problems, because hidden layers with nonlinear activation functions can be made to model nonlinear classifiers well. Here, the number of nodes in each hidden layer, as well as the number of hidden layers in each network, highly determines the accuracy of predictions. In this project, we explored the performance of both methods.

1.4 Convolutional Neural Network

Convolutional neural networks (CNNs) are artificial neural networks that work particularly well for image data. The structure of CNNs exploits strong local correlation in the inputs. This is done by enforcing local connectivity between neurons of adjacent layers. The inputs of a hidden unit at

layer n are some locally-connected subset of the units of the previous layer $n-1$, done in such a way that the input to layer n represent some tile of the units of layer $n-1$, with overlapping tiles.

In addition, CNNs utilize shared parameterizations (weight vector and bias) across layers. By constraining the same types of weights across layers, essentially replicating units across layers, allows for features to be detected regardless to their position in the initial input, making the network much more robust to real world image data. Constraining multiple weights also reduces the number of free parameters, increasing learning efficiency.

2 Experimental Details

2.1 Datasets

For this project, we primarily used a dataset released by Kaggle as a part of an ICML-2013 workshop in representation learning. This dataset included 28,709 labeled training images, and two labeled test sets of 3,589 images. Faces were labeled with one of 7 labels: 0 (Angry), 1 (Disgust), 2 (Fear), 3 (Happy), 4 (Sad), 5 (Surprised), and 6 (Neutral).

2.2 PCA

We used SciPy's implementation of PCA to extract eigenfaces from our data. In our experiments, we optimized the number of principal components k we took as our eigenface basis. Our features were then the orthogonal projection of each image onto this basis. When performing PCA, we added whitening to ensure that eigenfaces were uncorrelated.

2.3 Facial Landmarks

We used the open source implementation of [3], DLib, to detect facial landmarks in our images. We detected 68 notable points for each facial image, which we then used as features for our classification. The landmarks found for each image are shown in Figure 1.

2.4 Multiclass Classification

We used SVM with a radial basis kernel to classify our manually extracted features. We used SciPy's implementation of SVM for these purposes To optimize the performance of SVM, we performed grid search cross-validation on the training and validation data to optimize the free parameters C and γ in our model.

We also used neural networks with one to two hidden layers to classify our features. We used Google's TensorFlow neural network library for these purposes. For both one-layer and two-layer neural networks, we performed model selection on the number of hidden nodes and learning rate to optimize the final performance.

Figure 1: The first 12 test images with labeled facial landmarks as generated from DLib's shape recognizer.

2.5 Convolutional Neural Network

To implement our convolutional neural network, we used Google's TensorFlow neural network library. Our model follows the architecture described in [4], with a few differences in the top few layers. In particular, we use fully connected layers instead of locally connected layers. The model described in [4] was designed for ImageNet's object classification, but it translated well to our problem of classifying facial expressions.

2.5.1 Network Structure

We implemented a deep CNN with five layers, which we will refer to as `conv1`, `conv2`, `local3`, `local4`, and `softmax_linear`. This structure is illustrated in Figure 2.

The two convolution layers, `conv1` and `conv2`, divide the image into overlapping tiles and use those as inputs to the next layer. That is, the output can be written as

$$y(i,j) = \sum_{di,dj} x(s_1 i + di, s_2 j + dj) f(di, dj)$$

where $y(\cdot,\cdot)$ is the output, di and dj are changes which range over the tile size, s_1 and s_2 are strides between convolution tiles across each dimension, $x(\cdot,\cdot)$ is the input, and $f(\cdot,\cdot)$ are filters, i.e. the weights for the layer. For our model, we used 5 by 5 pixel tiles with 1 pixel strides. We did not regularize these convolution layers.

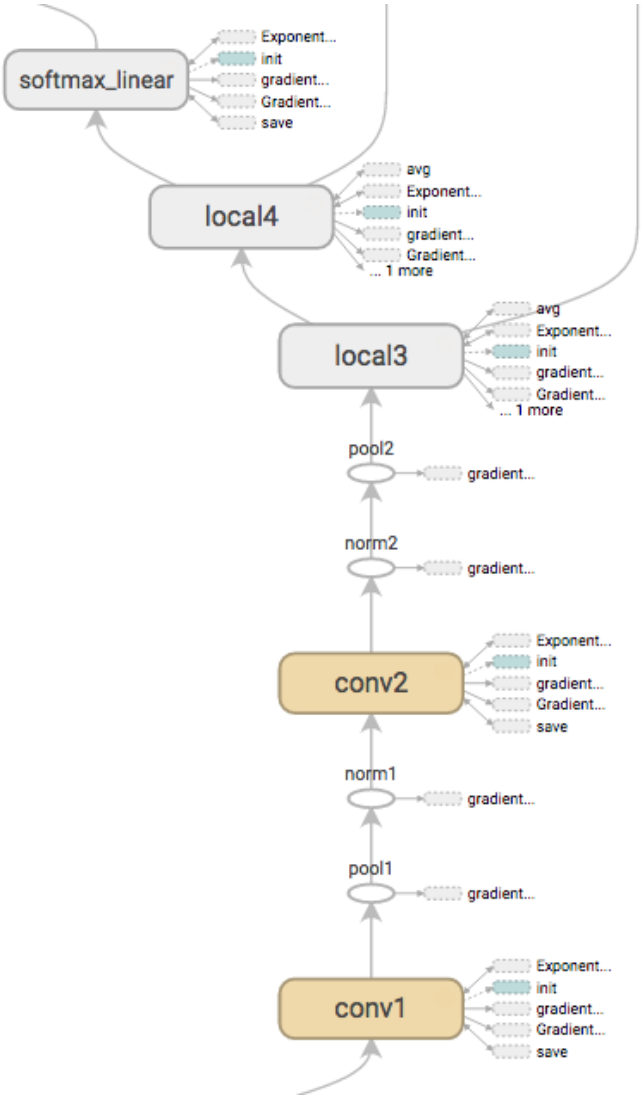

Figure 2: TensorFlow computation graph for the neural network in the CNN we implemented, illustrating each of the layers and computation done in between each layer.

As illustrated in Figure 2, after each convolution layer, we performed a `pool` operation and a `norm` operation. The max pooling operation takes the maxima among input tiles. That is,

$$y(i,j) = \max_{di,dj} x(s_1 i + di, s_2 j + dj)$$

Note that this is not a layer because there are no weights. The `norm` operation is a local response normalization response, where each input is normalized by dividing it by the squared sum of inputs within a certain radius of it. This technique is described in more detail in [4], and serves as form of "brightness normalization" on the input.

The `local3` and `local4` are fully connected layers with rectified linear activation $f(x) = \max(x, 0)$. We regularize both of these layers with the L2-loss of weights because these layers are fully connected and very prone to overfitting.

Finally, the `softmax_linear` layer is the linear transformation layer that produces the unnormalized logits used to predict the class of an input.

2.5.2 Training

The loss function we minimized was primarily the cross entropy between the multinoulli distribution, formed by the normalized logits outputted by our CNN, and the probability distribution that is 1 for the correct class and 0 for all other classes. The expression for cross entropy between two probability distributions p, q is

$$H(p,q) = E_p[-\log q] = -\sum_x p(x) \log q(x)$$

which measures the divergence between the two discrete probability distributions. In addition, we added standard weight decay, i.e. L2-loss, for the weights of the `local3` and `local4` layers for weight regularization.

For training, we fed the model randomly distorted training examples (see subsection 2.5.3) in randomly shuffled batches of 128 samples each. For each of these examples, we determined our loss based on our current inference. After each batch, we performed back propagation using gradient descent to learn the weights, and we called each time we finished a batch a step.

One technique which improved model accuracy by 3% when implemented was using moving averages of the learned variables. During training, we calculate the moving averages of all learned variables, and during model evaluation, we substitute all learned parameters with the moving averages instead.

2.5.3 Minimizing Overfitting

Since the size of our training set was 28,709 labeled examples, and our neural net needed to go through many more than

28,000 examples to achieve convergence, we employed a few techniques to ensure that our convolutional neural net did not overfit to the training set. This is also a risk because our top layers were fully connected, which means that our model could easily overfit using them.

We first implemented learning rate decay on our convolutional neural net, so that the learning rate decreased as it had been trained through more examples. We used exponential decay, so that that the learning rate decayed by a factor of 0.1 after training through $1,200,000$ examples, and we had it decay in a step change manner as is visible in Figure 8. We found that the step change learning rate decay worked better than a continuous exponential decay. This could be because maintaining a high learning rate initially could ensure that the CNN was trained towards a good local optimum in fewer steps, whereas a steadily decreasing learning rate could limit the range of the neural network.

In addition, we implemented distortion of the images while training to artificially increase the size of our training set, and make our convolutional neural net more robust to slightly distorted inputs. We processed our images in a few different ways. We cropped our 48 by 48 pixel images to a 40 by 40 pixel box, centrally for model evaluation and randomly for training. Then we approximately whiten the photos, scaling the pixel values linearly so that they pixel values have a mean of 0 and a standard deviation of 1, essentially normalizing them. This ensures consistent inputs to our neural network.

For training in particular, we performed a few more distortions to artificially increase our training set. We randomly flipped the image from left to right, randomly distorted the image brightness, and randomly distorted the image contrast for each image. These changes to randomly distort the images greatly improved the performance of the model, raising the correct classification rate from 45.4% to 53.8% when initially implemented.

3 Results and Analysis

3.1 PCA

We used the first $k = 150$ principal components of the training data as our eigenface basis. We found that this many eigenfaces explained 95% of the variance seen in our training data. However, we found that the features extracted from PCA were not very powerful representations of the data. In Figure 3, we show the images obtained from PCA against the original images in the test set.

We then optimized RBF SVM on the new features, and found that the optimal set of parameters $C = 5$ and $\gamma = 0.005$ yielded a test accuracy of 41%. The model selection grid search can be seen in Figure 4. Here, we see that we indeed have converged upon at least a local minimum over the ranges

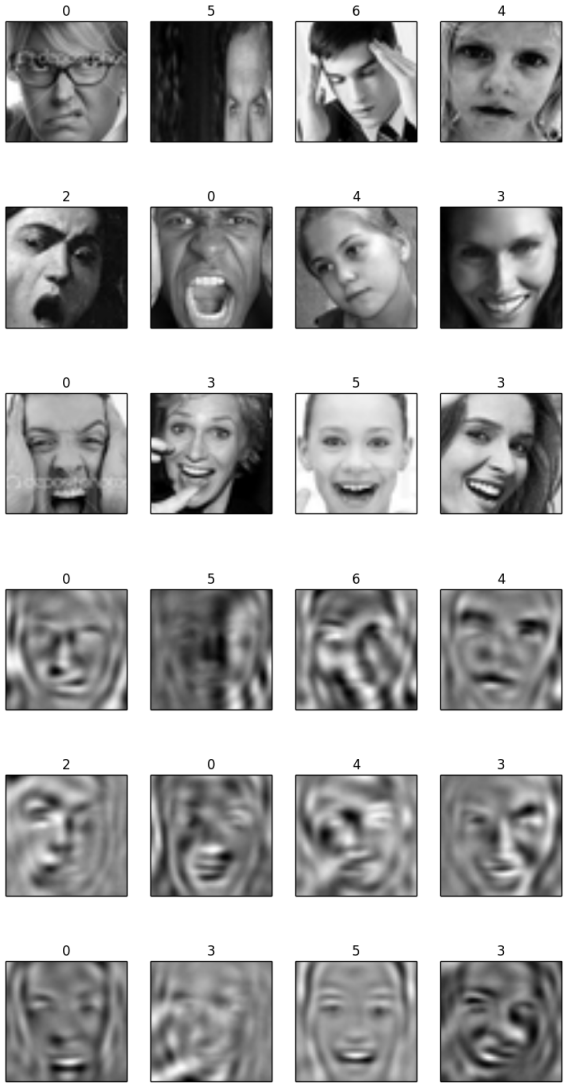

Figure 3: The first twelve images in the test set projected onto the 150-dimensional eigenface basis.

$C = \{ 1, 5, 10, 50, 100, 500, 1000, 5000\}$ and $\gamma = \{$5e-5, 0.0001, 0.0005, 0.001, 0.005, 0.01$\}$.

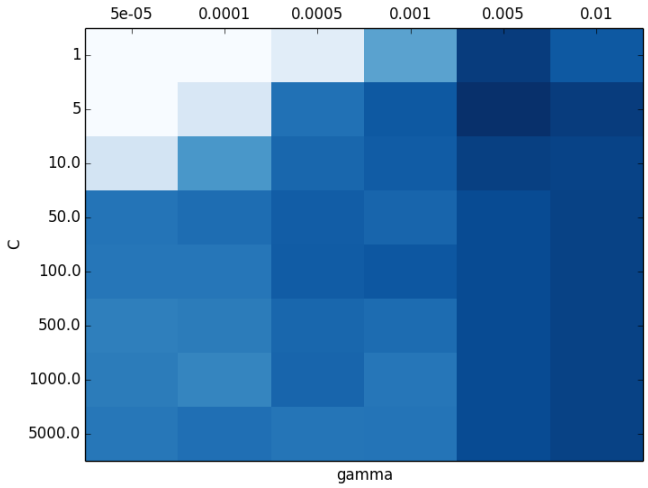

Figure 4: Cross validation on SVM with PCA features. Test accuracy for best parameters was 41%.

We also fed the features into our neural network for classification. We found that although different combinations of parameters yielded different convergence behaviors, those we tried all converged to at most accuracy of 40%. The convergence of a two-layer neural net with 80 hidden nodes is shown in Figure 5.

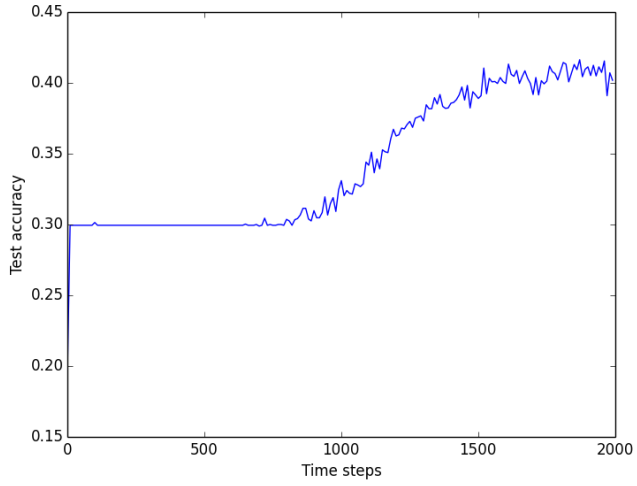

Figure 5: The convergence of a two-layer neural net with 80 hidden nodes for features extracted with PCA.

The mediocre classification accuracy on features extracted using PCA is unsurprising given the data. The features extracted were sensitive to the variance in the individual faces of the subjects, in addition to their facial expressions. Therefore our standard models of classification were unable to pick out the significant features to identify facial features. To address this problem we attempted using facial landmarks, which are more robust to variation among individual faces.

3.2 Facial Landmarks

The landmarks extracted from facial images are shown in Figure 1. For RBF SVM, we performed the same model selection grid search as described above, which can be seen in Figure 6. We found that the optimal set of parameters yielded a test accuracy of 52%, an improvement over the features extracted using PCA. Again we see that we have converged upon a minimum over the ranges $C = \{1, 5, 10, 50, 100, 500\}$ and $\gamma = \{$1e-7, 5e-7, 1e-6, 5e-6, 1e-5, 5e-5$\}$.

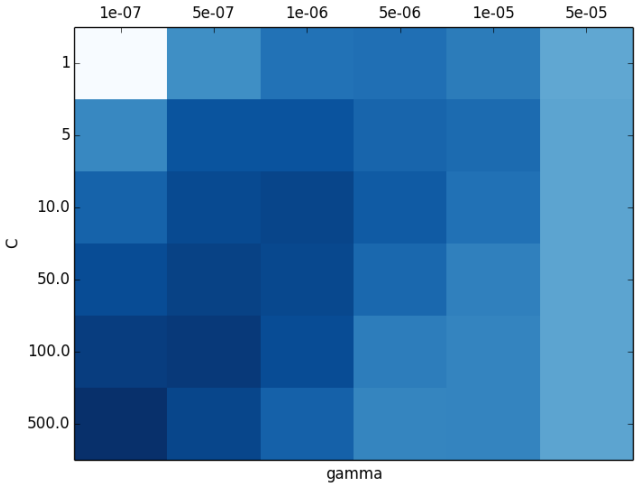

Figure 6: Cross validation on SVM with facial landmarks. Test accuracy for best parameters was 52%.

We again fed our features into a neural network for classification, and found that the models we tried all converged to the same value of around 50%, albeit with slightly different convergence behaviors. The convergence of a two-layer neural net with 100 hidden nodes is shown in Figure 7.

Here we note that although there was an increase in performance, the performance of both methods of classification was not impressive. In an attempt to improve the descriptiveness of our features, we normalized the landmark points with respect to the bounding box of the face, to minimize variance among individual feature dimensions. However, we found that this did not aid in test accuracy. We think this is because we did not extract enough landmark points to capture the fine facial changes between different expressions. Related to this is the quality of the landmark detector. However, such work is beyond the scope of this project. To improve performance, we experimented with convolution nets for both feature extraction and classification.

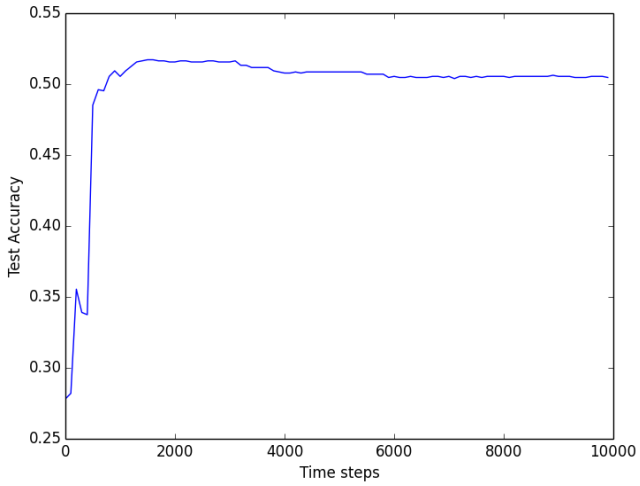

Figure 7: The convergence of a two-layer neural net with 100 hidden nodes for facial landmark features.

3.3 Convolutional Neural Network

We trained our model on Google Cloud Compute server with 4 CPUs and 15GB RAM for 20 hours using a training set of 28,709 labeled examples, and tested against a test set of 7,178 labeled examples. The model ran for 24,000 steps within those 20 hours, being fed a total of $3,072,000$ distorted training examples.

Our model achieved a correct classification rate of 65.2% on the 7-way classification problem, which is comparable to the 69% correct classification rate of the winning model on Kaggle. Our model correctly determines the correct label within the two highest logits with 82.3% accuracy, and determines the correct label within the three highest logits with 90.5% accuracy.

We can see how the total loss and the loss across the layers changed along with the learning rate in Figure 8. There are a few interesting patterns. First, the weight decay loss on the `local3` and `local4` layers fell very quickly (by the 5000th step) and did not decrease past that, while still decreasing the cross entropy loss. The `local4` loss even increased slightly.

In addition, the total loss, and in particular the cross entropy loss, dropped dramatically once the learning rate dropped according to the exponential learning rate decay schedule. This makes sense, since it allows the neural net to much more finely optimize the weights on the space it settled in with the larger learning rate. Finally, at the extremely small learning rate of 0.001, the loss converged at around 0.5, and any additional learning in the model at this point did not increase the classification success on the test set.

We can also see the change in the parameters of the model, weights and biases, and how they changed over time in Figure 9. We can see that the regularized layers `local3` and

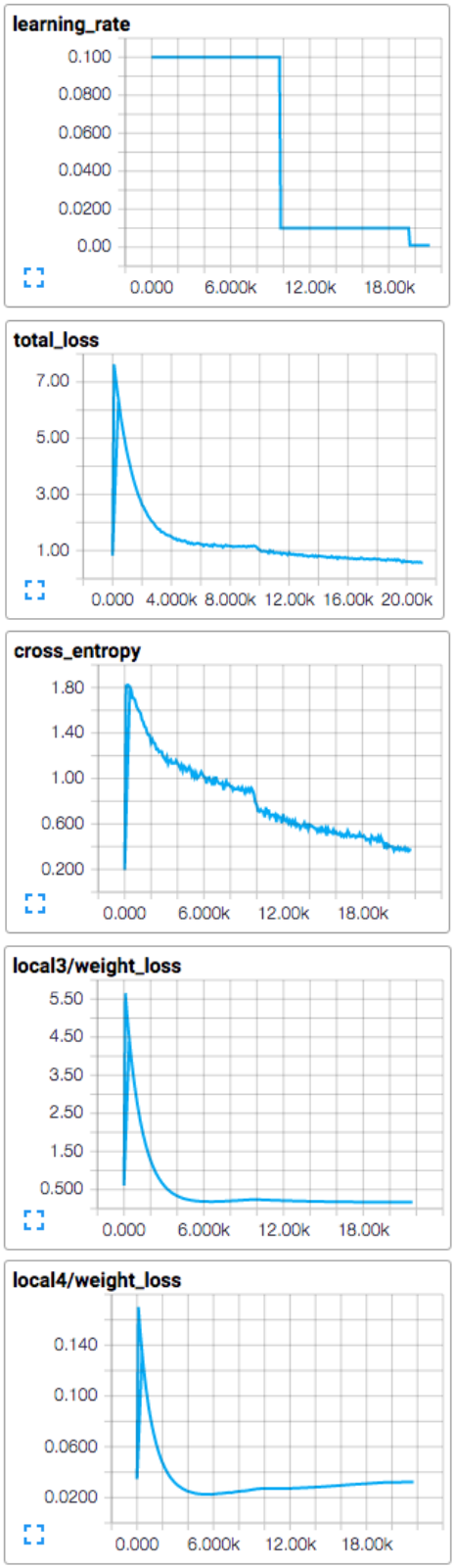

Figure 8: The learning rate, total loss, cross entropy loss, and regularized loss of layers `local3` and `local4` of the convolutional neural net over time. The x-axis is the step, and the y-axis is the value.

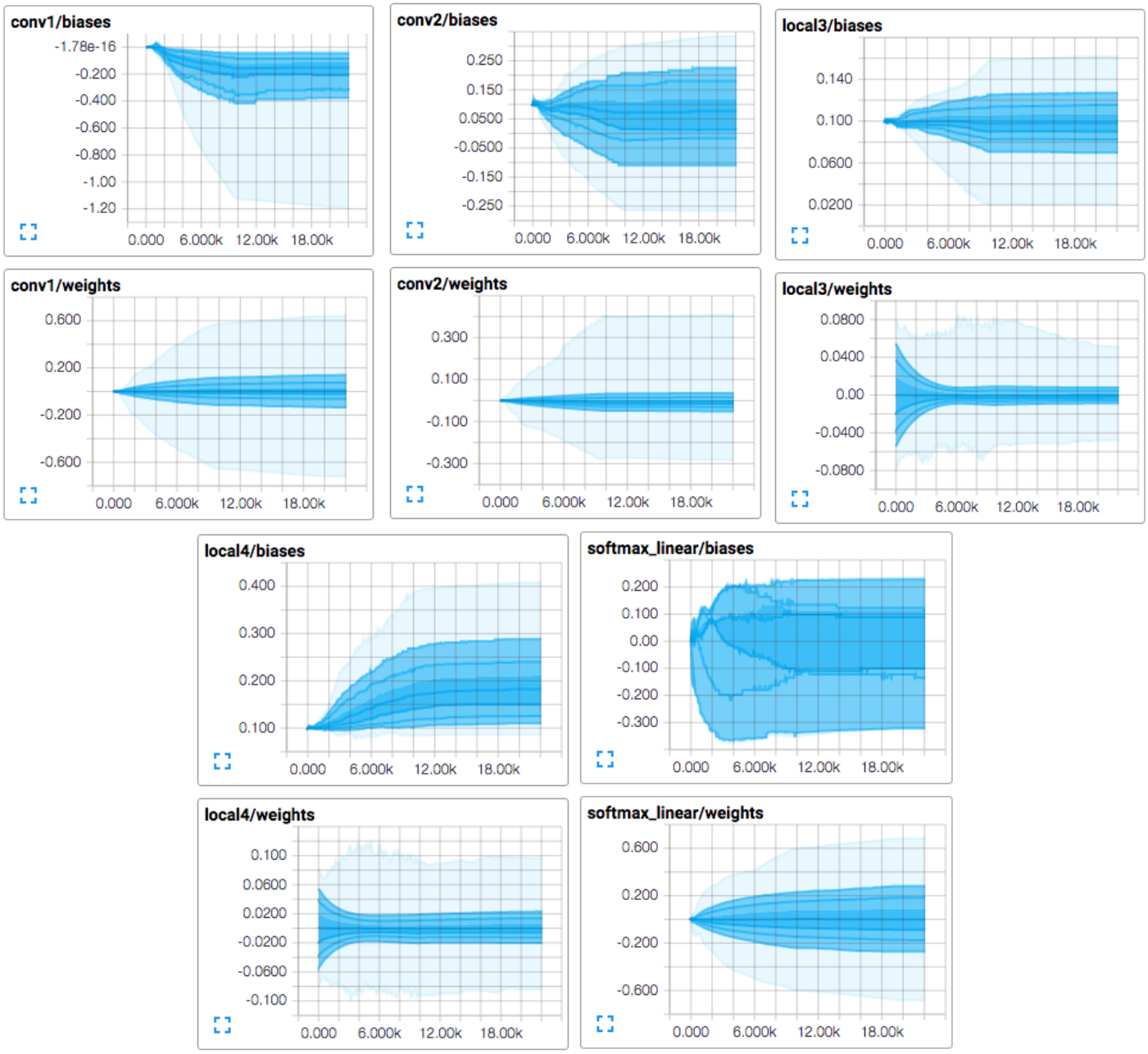

Figure 9: Histograms of the weights and biases of the layers of the CNN over time. The x-axis is the step, and the y-axis is the value.

`local4` had weights that quickly converged and stayed with approximately the same distribution. The weights on the non-regularized layers, i.e. `conv1`, `conv2`, and `softmax_linear`, mostly diverged with each step, but with the smaller learning rate they stopped increasing and maintained approximately the same distribution. Finally, it is interesting that the biases for layers `conv1` and `local4` were learned to be extremely skewed distributions. This indicates that the performance of the network could potentially be improved by initializing those weights at a more skewed distribution, decreasing the initial learning time.

One feature which could be implemented and would likely improve the performance of the algorithm is dropout, which could further prevent overfitting. This would randomly drop individual nodes with a certain probability p at each step, and would increase necessary training time by a factor $1/p$.

4 Conclusion

In this project we explored the classification of facial images into seven emotion classes: anger, disgust, fear, happy, sad, surprise, and neutral. We found that the representation of faces using eigenfaces from PCA was not robust or descriptive enough for successful classification; the variance between features was explained more by the differences in individual faces than by the differences in expressions. We obtained test accuracy of 41% using this method on our dataset. We found that the representation using facial landmarks extracted from subject faces was more succesful. However, the number of landmarks and the accuracy of the landmarks extracted were not particularly high, and we obtained a test accuracy of 52% using this method on our dataset.

When we applied a convolutional neural network with two convolutional layers and two regularized fully connected hidden layers, we found that test accuracy improved significantly. We also found that certain practices, such as randomly distorting, highlighting, darkening, cropping, and flipping images, during training helped build a more robust predictor. Our convolutional network obtained a test accuracy of 65% when applied to our dataset. We also found that 82% of the time, the true label was in the top 2 CNN predictions, and 91% of the time the true label was in the top 3 predictions. Further work on this problem could involve including dropout to further reduce overfitting during training and exploring different network structures.

References

[1] Lawrence, S.; Giles, L.; Tsoi, A. C.; Back, A. D. (1997) "Face Recognition: A Convolutional Neural-Network Approach" *Neural Networks, IEEE transactions on* 8 (1):98-113

[2] Matsugu, M.; Mori, K.; Mitari Y.; Kaneda Y. (2003) "Subject independent facial expression recognition with robust face detection using a convolutional neural network" *Neural Networks* 16 (5):555-559

[3] Kazemi, V.; Sullivan, J. (2014) "One Millisecond Face Alignment with an Ensemble of Regression Trees" *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*

[4] Krizhevsky, A.; Sutskever; Hinton, I.; Hinton, G. (2012) "ImageNet Classification with Deep Convolutional Neural Networks" *Advances in Neural Information Processing Systems* 25 (1):1097-1105

[5] Samal, A.; Iyengar, P. (1992) "Automatic Recognition and Analysis of Human Faces and Facial Expressions: A Survey" *Pattern Recognition* 25 (1):65-77

[6] Bartlett, M. S.; Littlewort, G.; Frank, M.; Lainscsek, C.; Fasal, I.; Movellan, J. (2005) "Recognizing Facial Expression: Machine Learning and Application to Spontaneous Behavior" *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*