Facial Keypoint Detection

Shayne Longpre, Ajay Sohmshetty
Stanford University
slongpre@stanford.edu, ajay14@stanford.edu

March 13, 2016

Abstract

This paper describes an approach to predicting keypoint positions on greyscale images of faces, as part of the Facial Detection (2016) Kaggle competition. Facial keypoints include centers and corners of the eyes, eyebrows, nose and mouth, among other facial features. Our methodology involves four steps to producing our output predictions. The first step involves a number of training time data augmentation techniques to expand the number of training examples, and generalizability of our model. For the second stage we will discuss a number of convolutional neural networks we trained, with the best architectures derived from former LeNet and VGG Net models. At test time, we will discuss our approach of again using data augmentation techniques to produce a composite of images from a single test image, allowing for averaged predictions. Lastly, we use a weighted ensemble of models to make our final keypoint predictions. We evaluate our different model architectures, with and without data augmentation techniques based on their Root Mean Squared Error scores they produce on the test set.

1 Introduction

1.1 Motivation

Our primary motivation for this project was our interest in applying deep learning to significant problems with relevant uses. The applications of this research are numerous and significant, including facial expression analysis, biometric or facial recognition, medical diagnosis of facial disfigurement and even tracking of line of sight. The idea that we were working towards solving an open and important problem with all these possible applications greatly inspired us. We specifically chose the Facial Keypoint Detection Kaggle competition because it gave us ample opportunity to experiment with a wide variety of approaches and neural net models to solve an otherwise straightforward problem of localization. The competition element also allowed us to benchmark our results against the greater community, and compare the effectiveness of our methods against alternatives. Lastly, we were motivated by the inherent challenges associated with the problem. Detecting facial keypoints is a challenging problem given variations in both facial features as well as image conditions. Facial features differ according to size, position, pose and expression, while image conditions vary with illumination and viewing angle. These abundant variations, in combination with the necessity for highly accurate coordinate predictions (eg the exact corner of an eye) lead us to believe it would a deep and interesting topic.

1.2 Problem Statement

This task is a currently active, designated kaggle competition [1], having started in May 2013 and expiring in December 2016. The problem is essentially to predict the near-exact coordinate locations of points on an image of a face. Given 15 facial points, there are 30 numerical values that we need to predict: a 2D

(x, y) coordinate representation for each facial point. The loss is calculated as the total root mean squared error (RMSE), an effective measure of the deviations in distances between the 15 real and predicted facial point coordinates.

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

The formula for Root Mean Squared Error is shown above. Here $n = 30$ given there are 30 dimensions to the output, y_i is the i-th dimension of the expected output and $\hat{y_i}$ is the i-th dimension of the predicted output. Our approach involves the use of deep learning and convolutional neural networks to predict the x and y coordinates of a given keypoint class in an image containing a single face. Therefore, as shown below, our input is a greyscale image from the dataset (96x96x1), and our output is 30 floating point values between 0.0 and 96.0, indicating where on the image the corresponding x and y coordinates are for a given keypoint feature. Below are example inputs (first row) and outputs (second row).

Figure 1

1.3 Dataset

Our dataset [1], provided by the University of Montreal Medical school, includes training data of 7049 grayscale $96x96x1$ images, with the labelled (x, y) coordinates for 15 facial keypoints. Facial keypoints include the right, left and centers of: eyes, eyebrows, nose, the upper and lower lips. Our task is to develop a model that can predict the specific locations of these facial keypoints on 1783 test set images, minimizing Root Mean Squared Error. Of the 7049 training images, there existed a distinct subset of 2140 which was completely and accurately labelled. We chose to discard the other images as they were generally lower quality, contained numerous errors and were incompletely labelled. Instead, to expand our dataset we used data augmentation, described further in Methods and Technical Approach.

2 Related Work

In the last decade a great deal of work has been done on this topic, of facial keypoint detection. Previously, it was common for detection to be based on local image features. For example, Liang et al. [2] searched faces at the component level using discriminative search algorithms and then 'adaptive boosting'. They used direction classifiers to help guide their discriminative algorithms. Others used random forest classifiers [3] or even SVMs [4] to detect facial components. Furthermore, cascading convolutional neural network architectures [8] and gabor filters [12] have been used to avoid local minima caused by ambiguity and data corruption in difficult image samples due to occlusions, large pose variations, and extreme lightings. From these papers it is clear that theses approaches are achieving solid results, but further research is required to refine component detection.
There are also a number of recent regression based approaches. For instance, Valstar et al. [7] used Markov Random Fields along with support vector regressors. It was found that the Markov Random Fields constrained the search space substantially, and the regressors learned the typical appearance of areas surrounding a given keypoint, allowing for rapid and robust detection despite variations in pose and expression. Dantone et al. [5] used local facial image patches to predict keypoints using conditional regression forests, which they found to outperform regression forests since they learn "the relations conditional to global face properties". Parkhi et al. [10] trained

a VGGNet [13] for the task of facial recognition, an architecture that secured first in the ImageNet Challenge in 2014. One last example is Caoe et al. [6] who used random ferns as regressors on the entire face image as input. Predicted shapes were represented as linear combinations of training shapes.

3 Technical Approach

3.1 Train Time Data Augmentation

We decided to use a variety of data augmentation techniques to (a) expand the size of our training data, and (b) help generalize our model. The most effective data augmentation techniques at train time that we ended up using were horizontal reflection, slight rotation and contrast reduction. Our procedure was executed as follows. For a given training image we would with probability 0.5 apply horizontal reflection. If the probability succeeded, we would have an 'augmented set' of 2 images: the original image, and the horizontally reflected image. Next, we would, with probability 0.5 for each of the images in the augmented set, apply the rotation transformation (as discussed below). Every time the probability succeeded, our 'augmented set' would be supplemented with an additional image. Lastly, on the images in the augmentation set we would apply the contrast reduction transformation with probability 0.5. With this stacking approach, for each training image there was a possibility of producing between 1 and $2^3 = 8$ images, inclusive (the final augmented set), which would all be added to the new training dataset as inputs to our models. The expected number of images in an augmented set from a single image is $1.5*1.5*1.5 = 3.375$. Therefore, on average, our training set of 2140 raw images was transformed into 7222.5 input images.

3.1.1 Horizontal Reflection (Mirror)

The first data augmentation technique is fairly straight forward. We need only reflect the image and its keypoint labels horizontally and then remap the keypoint labels to their new representations (left center eye becomes right center eye, and vice versa).

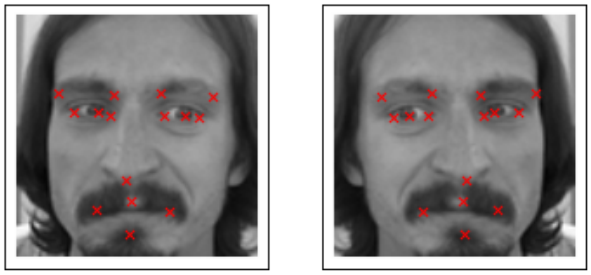

Figure 2

3.1.2 Rotation

For the second data augmentation we rotate the image clockwise or counterclockwise each with probability 0.5. The image pixel matrix (X) and labels are rotated using X•R, where R is the rotation matrix. The images are padded with their mean pixel value, along the edges where parts of the image were rotated out of bounds. After experimentation, we settled with hyperparameter $\theta = 5^o$.

$$R = \begin{matrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{matrix}$$

Figure 3

3.1.3 Contrast Jittering (Reduction)

For the third data augmentation we reduce the contrast of the greyscale image. This is done by applying the formula $CR(X)$ below, where X is the $96x96x1$ input and $mean(X)$ is the average pixel value in X. The idea is that pixel values are shifted slightly towards the images mean pixel value. The degree of shift is determined by hyperparameter δ

which we eventually set to 0.8 after experimentation. The greater the value of $1 - \delta$ the greater the shift towards the image's mean.

$$CR(X) = (\delta * X) + (1 - \delta) * mean(X)$$

Figure 4

In practice we found that each of these augmentations improved results individually, and even more so in combination. We could not discern significant differences from varying the probability of application for each as long as probabilities were between 0.4 and 0.6. The impact of contrast reduction increased substantially when $\delta = 0.8$, and was gradually less effective at higher values, and rapidly less effective at lower values. Once we found the optimal value of δ contrast jittering significantly improved results. This is most likely because the greyscale images within the dataset vary greatly in terms of contrast. Additionally, the mean pixel value varies quite significantly. The rotation angle hyperparameter produced generally good results for any angles below 12^o, but otherwise didn't vary greatly in performance.

3.2 Architectures

We first wanted to identify a "lower bound", of which to compare successive models to. As a baseline measure, we implemented a basic net consisting of a single fully-connected hidden layer, with 500 neurons. We used Adam parameter update method, with a learning rate of 0.03 annealed to 0.0001 across 2500 epochs. See Table 1 for more information.

Layer	Name	Size
0	input	1x96x96
1	hidden	500
2	output	30

Table 1: Baseline

Next, we implemented LeNet-style neural networks, consisting of alternating convolutional layers and max pooling layers, followed by fully connected hidden layers, with filter size $3x3$, pooling size of $2x2$ with stride 1. All hidden layers contained 500 neurons. We used Nesterov momentum parameter update scheme, with a learning rate of 0.01 and update momentum of 0.9. The largest LeNet we ran is described in Table 2, with Table 3 describing the same net in Table 2 with dropout added. We also implemented several VGG-style convolutional neural networks, consisting of several convolutional layers followed by single max pool layers, with fully connected layers and dropout. The largest VGGNet we implemented is described in Table 4.

Layer	Name	Size
0	input	1x96x96
1	conv2d1	16x94x94
2	maxpool2d2	16x47x47
3	conv2d3	32x45x45
4	maxpool2d4	32x22x22
5	conv2d5	64x20x20
6	maxpool2d6	64x10x10
7	conv2d7	128x8x8
8	maxpool2d8	128x4x4
9	conv2d9	256x2x2
10	maxpool2d10	256x1x1
11	dense11	500
12	dense12	500
13	dense13	500
14	dense14	500
15	output	30

Table 2: 5-LayerLeNet

Layer	Name	Size
0	input0	1x96x96
1	conv2d1	16x94x94
2	maxpool2d2	16x47x47
3	dropout3	16x47x47
4	conv2d4	32x45x45
5	maxpool2d5	32x22x22
6	dropout6	32x22x22
7	conv2d7	64x20x20
8	maxpool2d8	64x10x10
9	dropout9	64x10x10
10	conv2d10	128x8x8
11	maxpool2d11	128x4x4
12	dropout12	128x4x4
13	conv2d13	256x2x2
14	maxpool2d14	256x1x1
15	dropout15	256x1x1
16	dense16	1000
17	dense17	1000
18	dense18	1000
19	dense19	1000
20	output	30

Table 3: 5-LayerLeNetWithDropout

Layer	Name	Size
0	input	1x96x96
1	conv2d1	64x94x94
2	conv2d2	64x92x92
3	maxpool2d3	64x46x46
4	conv2d4	128x44x44
5	conv2d5	128x42x42
6	maxpool2d6	128x21x21
7	conv2d7	256x19x19
8	conv2d8	256x17x17
9	maxpool2d9	256x8x8
10	dense10	512
11	dropout11	512
12	dense12	512
13	dropout13	512
14	output	30

Table 4: SimpleVGGNet

3.3 Test Time Data Augmentation

At test time, we used the data augmentation that were applied to the training data to improve our models predictions. For a given test image we applied the following transformations with $probability = 1.0$: (a) horizontal reflection (mirror), (b) 5^o clockwise rotation, (c) 5^o counter-clockwise rotation, (d) $\delta = 0.8$ contrast reduction, (e) $\delta = 1.1$ contrast increase. The contrast increase uses the same formula $CR(X)$ described in section 2.1.3, but with a $delta \geq 1.0$ the effect is to increase contrast rather than reduce it. It is essential that the resulting values be clipped between 0.0 and 96.0 in case certain pixels deviate beyond the bounds. Using the 5 augmented images and the original test time image (6 in total) we predict 6 sets of facial keypoints, one for each augmented image in the set. After applying augmentation reversal transformations on the predicted coordinates, we have 6 sets of facial keypoint predictions for the original image. For instance, a reversal transformation on a 5^o clockwise rotated image would be to rotate the predicted keypoints 5^o counterclockwise. For the horizontally reflected image, to reverse the transformation we must reflect it back horizontally and then remap the keypoint labels. With the final 6 facial keypoint predictions for the original image we can average them to produce a final prediction for each point.

3.4 Ensemble

At test time, we used an ensemble model consisting of our three best performing models (LeNet, LeNetWithDropout, and SimpleVGGNet), which produced predictions as the weighted average of the regressions. We adjusted the weightage of each model based on its validation accuracy, weighting our best performing model highest.

4 Results

Our preliminary results are shown below. For each model we spent a great deal of time tuning the hyperparameters and experimenting with different update rules and number of epochs to achieve the

best possible RMSE given the model architecture. Specifically, our learning rate was the most crucial metric to correctly and appropriately tune. A large learning rate caused loss to explode, whereas too small a learning rate resulted in a stagnant loss curve. We also tried adding Rectified Linear Units (ReLU) and batch normalization layers, but acheived a validation loss of around our baseline, initially implying adding these required additional hyperparameter tweaking. However, after performing grid search over 100 epochs to find the optimal hyperparameters, we could not acheive a validation accuracy better than our baseline. Reducing the hidden dimension count to less than 500 for fully connected layers also resulted in a much higher validation loss, around our baseline's. We also altered the number of filters at each convolutional layer, and found that increasing powers of two filter sizes were the optimal choice, with a higher number of filters count not significantly improving loss but linearly impacting the runtime. After hyperparameter tuning, we found that a learning rate of 0.1 to 0.05 works best, with Nesterov's momentum update working most effectively. Annealing the learning rate was very unpredictable and ungeneralizable, so we reserved its use for just the baseline. See Table 5 for our results on the various models we tested, Table 6 for the Kaggle submissions we made, Figures 5-9 for our plotted losses across epochs (TLoss = Training Loss in $\times 10^{-3}$, VLoss = Validation Loss in $\times 10^{-3}$, VRMSE = Validation RMSE, with "no aug" representing the models in which we did not use data augmentation.).

Model	TLoss	VLoss	VRMSE
Baseline (no aug)	4.43	4.19	3.107
LeNet-2 (no aug)	0.40	1.50	1.856
LeNet-3 (no aug)	0.66	1.42	1.809
LeNet-3	0.68	1.01	1.523
LeNet-4 w/Dropout	2.60	3.13	2.685
LeNet-5	0.23	0.79	1.349
LeNet-5 w/Dropout	2.41	2.63	2.462
SimpleVGG	2.62	2.03	2.163

Table 5

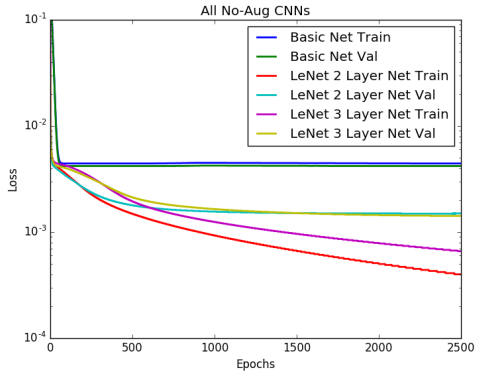

Figure 5

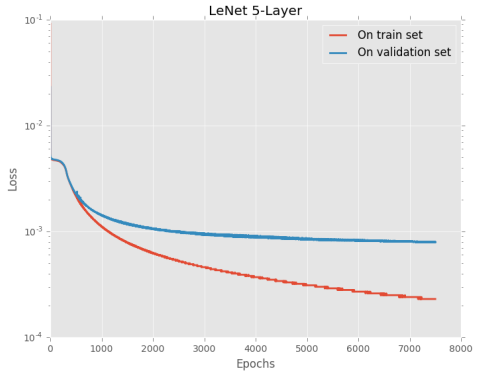

Figure 6

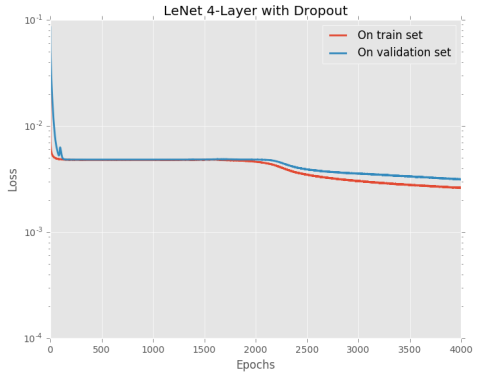

Figure 7

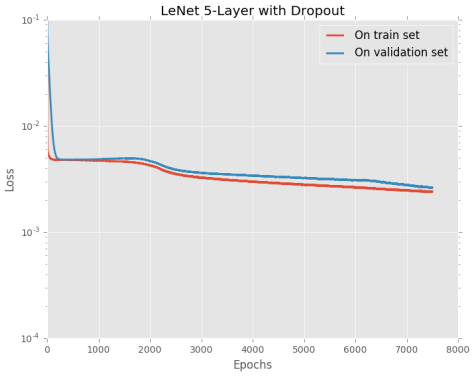

Figure 8

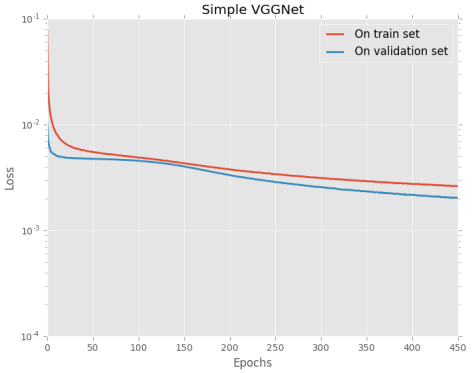

Figure 9

Model	Kaggle MSE
LeNet-5 w/o test time aug	3.150
LeNet-5	3.089
Ensemble (LeNet-5 + dropout)	3.347
Ensemble (LeNet-5 + LeNet-3)	3.104
Ensemble (LeNet-5 + VGG)	3.300

Table 6

5 Analysis

As we can see from Table 5, our best pAs we can see from Table 5, our best performing model is our 5-layer LeNet, without dropout, and with data augmentation enabled. This model, however, shows signs of overfitting, with a large gap between train and validation errors. This inspired our motivation to add regularization, adding dropout of p=0.5 . However, as we observe in Figure 8, while this brought the train and validation error curves closer together, overall the loss was higher than without dropout, at .00263 vs .00079 validation loss. On the other hand, training time data augmentation had a significant beneficial impact, clearly illustrated by our LeNet-3. Without data augmentation, we obtain a validation RMSE of 1.856, compared to a validation RMSE of 1.523 with. We also see a trend of deeper convolutional networks with more learnable parameters resulting in better errors, and a lower loss. However, while this trend is consistent within architectures, it is not observed between entirely different architectures. Our Simple-VGG-Net, with 14 layers, was by far the most difficult to train, with the most number of learnable parameters at 9,811,422 and each epoch taking 300 seconds on AWS GPU optimized instances. Nonetheless, this large architecture was not able to outperform any of our LeNet architectures. It is worth noting, however, that it may not have reached full convergence, judging by Figure 9, as we still see a slight downward gradient. Given additional time, we would have liked to run VGGNet over more epochs. Interestingly, we see the validation loss for VGGNet as significantly lower than the train set, implying our model has good generalizability, likely due to the addition of our dropout layers. Based on Table 6, we see that our test time data augmentation routine had a positive impact on our MSE, an improvement on our best neural network from 3.150 to 3.089. Out of the 157 submission entries, our MSE of 3.089 ranks us at 34. Contrary to our expectations, the ensemble models did not outperform LeNet-5. This is perhaps because the gap between the validation accuracies of LeNet-5 and the other models is too large for ensembling to be effective.

6 Conclusion and Future Work

We found that train time data augmentation techniques, especially low angle reflection and contrast reduction significantly improved the generalizability

and RMSE of our models. Additionally, test time averaged data augmentation predictions gave us a small boost. We speculate that the model ensembles would have had a positive impact had our best models performed relatively similarly on the validation RMSE. Our best architectures were deep LeNet models without dropout, though with greater computing resources VGGNet may have been able to achieve similar or lower validation scores with less overfitting. Our future work will involve using the VGG Face pretrained model, fine tuned over the last few layers. This could also be a contender for an effective ensemble model. Additionally, we would consider segmenting our best model into more specialized models that focus only on a subset of the facial keypoints. For example, one model might specialize specifically on eye features, another specifically on nose features, and so on.

References

[1] Facial Keypoint Detection Competition. Kaggle, 7 May 2013. Web. 31 Dec. 2016. ¡https://www.kaggle.com/c/facial-keypoints-detection¿. 1.2, 1.3

[2] Liang, Lin, et al. "Face alignment via component-based discriminative search." Computer Vision–ECCV 2008. Springer Berlin Heidelberg, 2008. 72-85.

[3] Amberg, Brian, and Thomas Vetter. "Optimal landmark detection using shape models and branch and bound." Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011.

[4] Belhumeur, Peter N., et al. "Localizing parts of faces using a consensus of exemplars." Pattern Analysis and Machine Intelligence, IEEE Transactions on 35.12 (2013): 2930-2940.

[5] M. Dantone, J. Gall, G. Fanelli, and L. J. V. Gool. Real-time facial feature detection using conditional regression forests. In Proc. CVPR, 2012.

[6] D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In Proc. CVPR, 2012

[7] M. Valstar, B. Martinez, X. Binefa, and M. Pantic. Facial point detection using boosted regression and graph models. In Proc. CVPR, 2010.

[8] Sun, Yi, Xiaogang Wang, and Xiaoou Tang. "Deep convolutional network cascade for facial point detection." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2013.

[9] Ciresan, Dan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification." Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012.

[10] Parkhi, Omkar M., Andrea Vedaldi, and Andrew Zisserman. "Deep face recognition." Proceedings of the British Machine Vision 1.3 (2015): 6.

[11] Nouri, Daniel. 2015. Github. Kaggle Facial Keypoints Detection tutorial. Available from https://github.com/dnouri/kfkd-tutorial/blob/master/kfkd.py

[12] Serre, Thomas, Lior Wolf, and Tomaso Poggio. "Object recognition with features inspired by visual cortex." Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 2. IEEE, 2005.

[13] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).