Facial Keypoints Detection

Hao Zhang (hzhang10, 61526092)
Jia Chen (jiac5, 35724702)
Nitin Agarwal (agarwal,84246130)

1. Introduction

The objective of the project is to detect the location of keypoints on face images. Actually, detection of facial points is critical for a lot of applications such as face detection, facial expression analysis, etc. Original data and problem description can be found from Kaggle (http://www.kaggle.com/c/facial-keypoints-detection).

For training we have 7049 grayscale images of equal size (96x96). For each training image, 15 keypoints are provided with both x and y coordinates, allowed omission for portion keypoints. Hence mostly there are 30 or fewer target labels for each training image. Test data set has 1783 images and we need to predict the keypoint coordinates for the test images to obtain the Kaggle score (using RMSE).

Basically, it is a large-scale (9216 features) multivariate (30 labels) machine-learning problem and we have attempted a broad range of approaches, mainly including linear regression with CV method, SVMs, feed-forward neural network as well as convolution neural network. The experiment result shows the superiority of neural network model and we obtain best RMSE score of 3.21 using convolutional neural network for Kaggle dataset.

2. Linear Regression

2.1 Regression Challenges

To predict keypoint coordinates on human faces, an intuitive idea is that we can build a simple linear regression model with 96×96 features input and 15×2 labels for facial keypoints' coordinates. A simple attempt soon reveals the disability of this method: 1) Overfitting: Our linear regression model works quite well with the training image – the mean square error (MSE) for training data set is mere 8.76. However, when we apply the linear model for test data set, the predicted coordinates are sometimes out of range. 2) High dimension issues: It is intuitive that we can subsample a portion of pixels to reduce the dimension, which may help address overfitting. However, we do not know how many features are sufficient and what features are the best candidates.

2.2 Feature Selection

Motivated by the above observations, heuristic feature selection seems to be an appealing choice. One might think of stepwise methods. Yet we notice that stepwise method is normally chosen only if there is no underlying theory on which to base the feature selection [Efroymson, 1960]. Hence taking the advantage of a good heuristic for feature selection rather than random sampling seems a better choice. More than a pure machine learning problem, the facial keypoint regression actually has a directly perceived heuristic for feature selection, i.e., we can focus on the pixels features describing eyes, nose and mouth. We apply a robust algorithm called "Histogram of Oriented Gradients" to crop those pixel patches as shown in Figure 2.2 [Navneet Dalal, 2005]. We write the CV code by ourselves and for more implementation details please refer Appendix. B.

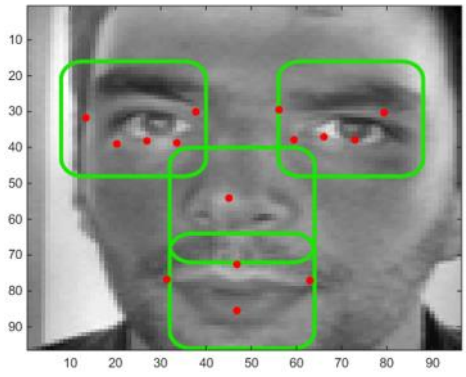

Fig 2.1 Typical prediction results by linear regression with HOG

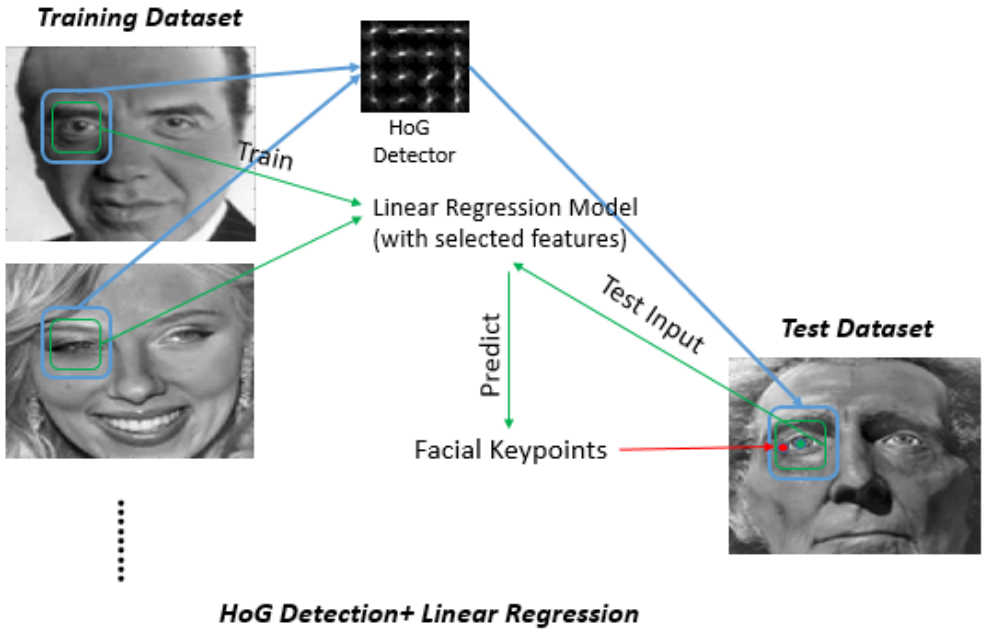

Fig 2.2 Algorithm diagram and typical prediction result

2.3 Complexity Control

In our implementation, we create a H × W window centered at left-eye, right-eye, nose and mouth respectively and select the pixel located inside these windows as input features to feed linear regression learner. Apparently, we can control the complexity of our model by varying the size of feature window. Increasing H or W will boost the complexity of our model and its predicted results behaves more similarly to that of learning the whole image. Figure 2.3 indicates the varying trends between MSE and the values of H, W respectively, for training and validation data set.

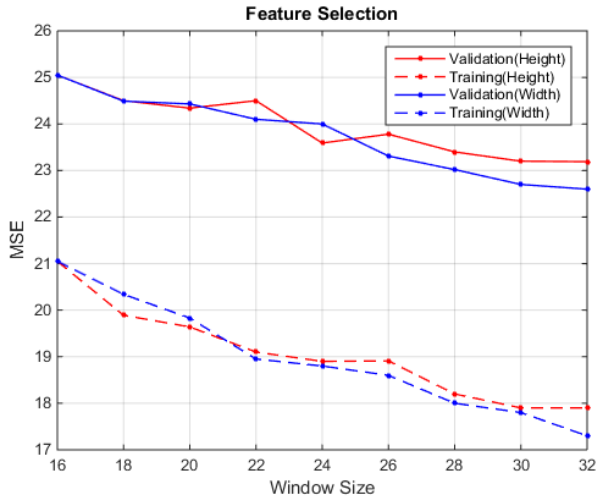

Fig 2.3 MSE results with different features size

Both validation and training MSE decreases with the increasing window size. It is an indicator that our model is still underfitting. However, it is not practical to incorporate more features due to the limitation of our algorithm (windows might exceed the range of indexes). In the next part, we will apply SVMs to automatically achieve feature selections. By perception, the best H and W values are 32, using which we obtain a RMSE score 4.6367 for Kaggle test data set. In fact, we can train our HOG detectors by learning a collection of templates, which can be seen as a clustering process. However, the implementation for this extended function is not trivial and we skip it here.

3. Support Vector Machine

SVMs (Support Vector Machines) are a useful technique for data classification and regression. In our implementation of SVM, we use the popular LIBSVM libraries [Chih-Jen Lin, 2003]. For this specific problem, we train a series of models for each facial keypoint so that we can invoke the API directly. Actually, facial keypoints detection is a multivariate regression problem, since there exist certain constraints among the labels (coordinates) of facial keypoints. To take it into consideration, more sophisticated models, like SVM^{struct} [Joachims, 2008], are required and we skip them here. Another point worth mentions is that scaling before applying SVM is very important. Although for our problem the range of each features is [0,255], our SVM can still benefits from scaling because kernel values usually depend on the inner products of feature vectors and large attribute values might cause numerical problems. In experiment, we scale our features to [0,1] instead of [-1,1], since there are quite a few zero entries in our data set, [0,1]-scaling can maintain the sparsity of input data and hence may save significant time.

3.1 Kernel Selection

3.1.1 Linear Kernel

In general, linear kernel is a reasonable first attempt if the number of feature is large and has the similar size of instances, we may not need to map data to a higher dimensional space. Using the linear kernel may already be good enough and we only need to search for the regularization parameter C. Increasing cost value (or named penalty parameter) C causes closer fitting to training data. For linear kernel, the best C is 0.1 and the corresponding model reduces the MSE from 10.663 (with default parameter) to 10.454. Also this model achieves a Kaggle score of 4.35512. Since the internal function for parameter optimization can only find a region instead of accurate values, in the RBF kernel section, we will show how we tune our models to obtain the best performance.

3.1.2 Polynomial

Now we consider the kernel function as polynomial. The most crucial parameter that allows us to control the complexity is the degree of the polynomial function. For the SVM learner, we vary the degree setting and obtain the following table indicating the relation between MSE and degree setting.

Table 3.1 Results of polynomial kernel with different degrees

Polynomial Degree	1	2	3	4	5
Cross-Validation MSE	10.8865	11.1539	11.4121	11.5706	11. 651

It is surprising that the SVM with polynomial kernel function overfits the data from degree 1, which implicitly validates our previous feature selection method since data overfitting may already exist for linear SVM. Also we notice that due to the implementation difference in linear kernel and polynomial kernel with degree 1, their MSE varies slightly and the former one obtain better result. The polynomial kernel with degree 1 achieves Kaggle score of 4.37543.

3.1.3 Radial Basis Kernel and Parameter Tuning

This kernel nonlinearly maps the samples into a higher dimensional space so it, unlike the linear kernel, is able to handle the case when the relation between class labels and features is nonlinear. Furthermore, the linear kernel is actually a special case of RBF [Keerthi and Lin 2003] since the linear kernel with a penalty parameter $\tilde{C}$ has the same performance as RBF kernel with some parameters(C, γ). Besides, the sigmoid kernel also behaves similarly when RBF is set with certain parameters [Lin and Lin, 2003]. As a consequence, the RBF SVM with good configuration is supposed to have no worse performance than other kernel SVMs.

As we mentioned before, C is the cost value which represents the penalty associated with error larger than epsilon. Increasing cost value causes closer fitting to training data. For kernel parameter γ, it controls the shape of the separating hyperplane. Increasing gamma usually increases number of support vectors. Hence, if our RBF SVM

underfits data set, we should increase cost value and decrease gamma. For overfitting situation, we tune the parameters in the opposite direction.

To localize the rough region of optimal parameters, we adopt a "grid-search" on C and γ using cross-validation. Although there are several advanced methods, which can save computation cost by approximating the cross-validation rate, there are two motivations for preference to grid-search approach: guaranteed optimality and parallelizability. By grid-search strategy, we can obtain the following "accuracy" contour map:

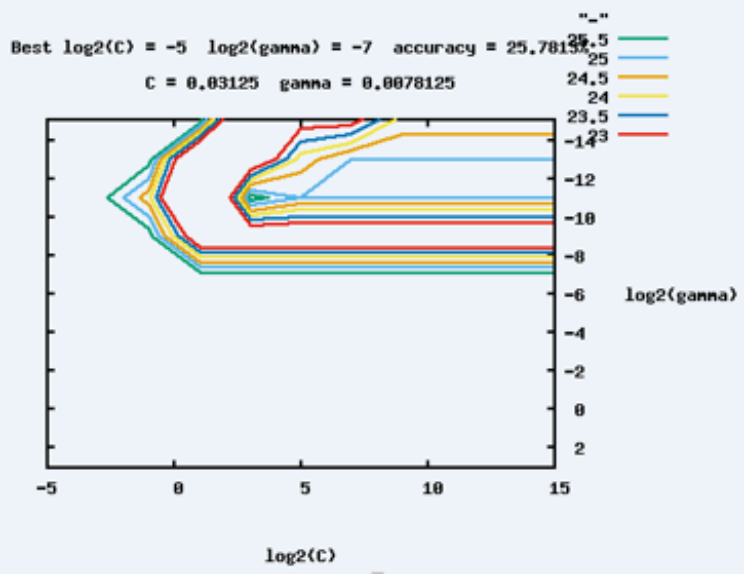

Fig 3.1 Contour map for classification accuracy

Since grid.py can only draw the contour map for integer labels, we have to round facial keypoints' coordinates for classification instead of regression and that is the reason why the grid search uses validation accuracy as the evaluation. Yet It does not matter much because when we invoke the grid.py with real coordinate, we obtain the similar results for C and γ (C=0.03125, γ=0.0078125) and the corresponding validation MSE is 8.99968.

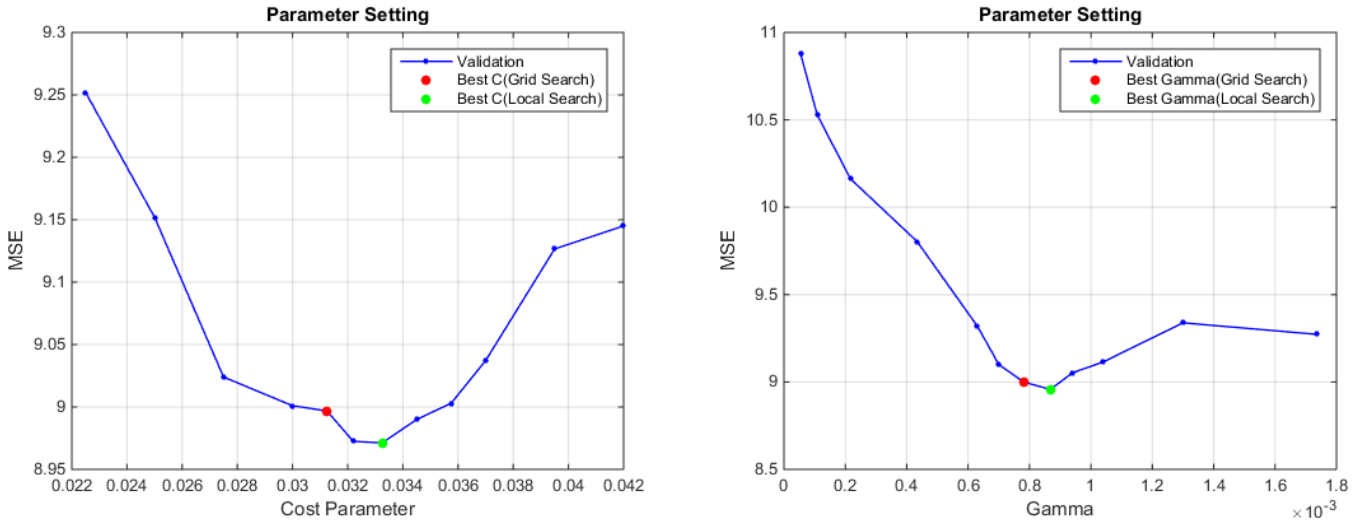

Fig 3.2 Cost and gamma values optimization

To find the accurate optimal values for C and γ we sample more points for C and γ in the optimal region. We first tune C (optimal value 0.03325), which is directly correlated with the model's sensitivity to training data. Then we fix it and search for best configuration of γ (optimal value 0.00868). With best parameter configuration, the RBF SVM obtain score 4.31254 for Kaggle data set.

3.2 Ensemble SVMs

In this section, we are trying to attempt different kernel function with various setting for better complexity control.

3.2.1 SVMs with Bagging

We apply bagging strategy to avoid overfitting. In this strategy, we train our models with partial data set and apparently there are two approaches to achieve this.

(1) Bagging on subsets of training data sample: In the training data set provided by Kaggle, there are 2140 images that have labels values for every facial keypoints. Since it is a time-consuming operation for large-scale SVMs, we only train 6 separate models each of which only utilized 1900 random images in the training data set. To combine their results for test data set, we simply average 6 models' predictions as final predicted facial keypoints. This strategy gets the score of 4.25613 on Kaggle test data.

(2) Bagging on different portion of image features: This feature selection idea is exactly what we have adopted in the previous method. Instead of only focusing on the pixels centered at the facial keypoints, each model now is trained on 8000 random pixels with replacement. The model based on this strategy gets RMSE score 4.23468

3.2.2 Mixed SVMs

In our experiment, we also combine SVMs that are trained by different kernels. Specifically, we average the prediction results of linear SVM and the bagging models. However, the performance of combined SVM does not benefit from this mixture, which only obtains score 4.26578 from Kaggle.

3.3 Summary of Results

In this chapter, we explore SVMs with different kernel functions as well as various parameter setting for complexity control. Linear SVMs with default setting achieve score around 4.5 and we reduce it to 4.25112 by trying a series of parameter configuration and generalized models. The Kaggle score for all our SVMs are summarized below

Model ID	Model configurations	Score on Kaggle (RMSE)
1	Linear Kernel SVM	4.35512
2	Polynomial Kernel SVM (degree 1)	4.37543
3	RBF Kernel SVM	4.31254
4	Bagging (6 models trained by different samples of data)	4.25613
5	Bagging (6 models trained by different features of data)	**4.23468**
6	Mixture of Models	4.26578

4. Feed-forward Neural Network

4.1 Architecture

There are several main types of neural network architecture such as feed-forward neural networks, recurrent neural networks, and symmetrically connected networks, etc. And in our project, we apply feed-forward neural networks, which is the commonest type of neural network in practical applications.

As introduced in [Bishop, 1995], given same number of units (with non-linear activation), a deeper architecture is more expressive than a shallow one. So a typical architecture of feed-forward neural networks contains at least 3 layers, the first layer is the input layer, and the last layer is the output. Between the input and output layers are the hidden layers, and if there are more than one hidden layer, the neural networks are called deep neural networks. In our project, each input image has 96 * 96 = 9216 pixels, so the input layer are composed by 9216 input units. And there are 15 facial keypoints we want to predict, so output layer are composed by 30 output units (x,y coordinates each in a unit). Hidden layers determine the complexity and expressiveness of our model, and how to determine its settings will be discussed in next section.

4.2 Determining Model Settings

In neural networks, the models' complexity is mainly determined by the number of hidden layers, the number of hidden units in each layer, and number of epochs (training iterations). In our project, we apply a state-of-the-art machine learning library Theano [http://deeplearning.net/software/theano/]. As Theano provides GPU acceleration, it is able to train one epoch for a single layer neural network with 100 hidden units in less than 0.1s. This enables us to experiment different settings to train various neural networks, and apply cross validation to determine which setting to use.

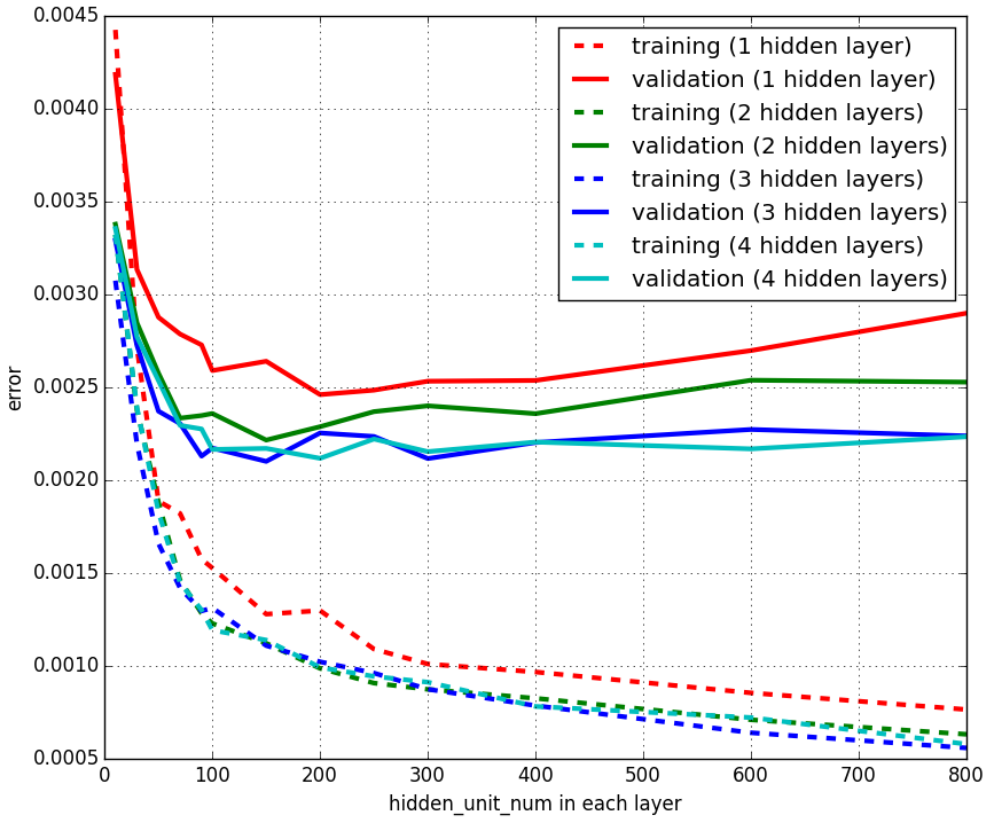

Fig 4.1 Training error and validation error under various numbers of hidden layers and hidden units

4.2.1 Choosing number of hidden layers and hidden units

Figure 4.1 shows training error and validation error with various numbers of hidden layer and hidden units inside the neural network. As shown in the figure, the model complexity increases along with the number of hidden units in each layer, and it is obvious that when the number grows beyond 300, the model is overfitted as the validation error starts to increase. And with more hidden layers, the validation error is generally smaller. We compare testing error for 3 models with 400 hidden units in each layer, and 1, 2 and 3 hidden layers respectively. On Kaggle test data set, their error scores (RMSE) are 3.82109, 3.85388, and 3.81652 respectively. This result shows that the impact of the number of hidden layer on testing performance is trivial, so we will generally adopt single hidden layer neural network to avoid unnecessary complexity.

4.2.2 Choosing number of epochs

Figure 4.2 shows the training error and validation error with various numbers of epochs. As the number of epochs increases, the training error decreases continuously, which indicates that the complexity of the model is increasing. The gap between the training and validation error often indicates the amount of overfitting. In our experiment, along with the growth of the number of epochs, the gap is increasing, which is a sign that the overfitting exists. So we choose a relatively small number of epochs 1000 to avoid unnecessary overfitting.

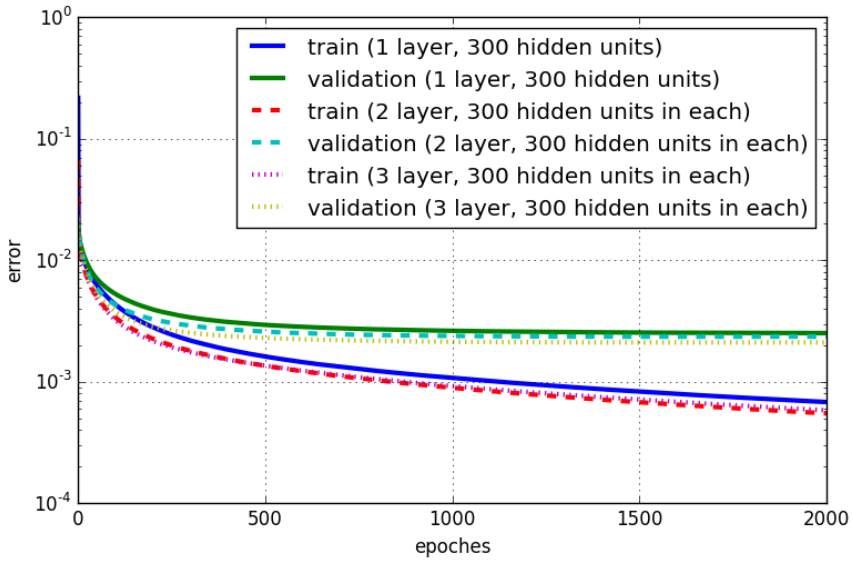

Fig 4.2 Training error and validation error under variable number of epochs

4.3 Generalization

In section 4.2, we have chosen model settings, which make the neural network complex enough to provide an adequate fit for the input data. But the more complex model we use, the more possible that the model is overfitted. So in this section, we will try to generalize our model to avoid such potential overfitting. Typical generalization methods include bagging, early stopping, regularization, and mixture of models, etc. In our project, we mainly apply bagging and mixture of models.

4.3.1 Bagging

We apply bagging strategy to prevent overfitting, which trains the model on different subset of the training data. In our experiment, we again apply the previous bagging strategies with slightly difference:

(1) Bagging on subsets of training data sample: in the training data set provided by Kaggle, we have 2140 labeled training images in total, thus we train 10 models, but we don't use all the images we have to train each model, instead of that, each model is trained on 1940 random images. And during testing stage, we use the average of all 10 models' prediction as result. This strategy gets RMSE score 3.67293 on Kaggle test data.

(2) Bagging on different parts of image: each of the images we are trying to learn has 96x96=9216 pixels, and we train 10 models, each model is trained on 8000 random pixels. Similar to the first strategy, during testing stage, we use the average of all 10 models' prediction as result, although before prediction, we need to reduce the dimension of each image according to the model. This strategy gets RMSE score 3.62655.

4.3.2 Mixture of Models

Averaging the predictions of many different models is another good way to reduce overfitting. And it helps most when the models make very different predictions. We design two mixture models as below:

(1) Mixture of neural networks with different number of hidden layers. As we trained 3 neural networks with 1, 2 and 3 hidden layers respectively, the mixture model takes the average of the 3 models' results as its output. And this strategy gets RMSE score 3.62702, which is better than each individual neural network in the mixture model.

(2) Mixture of model in strategy (1) and the second bagging model. As these two model perform the best so far, we further improve them by mix these two together. And results prove this is a reasonable strategy, its RMSE score 3.58884 is the best score we get in all of the models we try.

4.4 Summary of Results

In this chapter, we explore to apply feed-forward neural network to predict the feature points locations. Basic setting of neural network achieved RMSE score around 3.8, and we explored to apply a series of generalization methods to reduce the amount of overfitting. With bagging and mixture of models, the testing score was improved to 3.58884.

Table 4.1 Results of different model settings and generalization methods

Model ID	Model configurations	Score on Kaggle (RMSE)
1	1 hidden layer, 300 hidden units	3.82109
2	2 hidden layers, 300 hidden units in each layer	3.85388
3	3 hidden layers, 300 hidden units in each layer	3.81652
4	Mixture of model 1,2,3	3.62702
5	Bagging (10 models trained by different subsets of training data)	3.67293
6	Bagging (10 models training by different parts of image)	3.62655
7	Mixture of model 4,6	**3.58884**

5. Convolutional Neural Network

5.1. Architecture

Convolutional neural networks (CNN) were first introduced in late 1990's [LeCun, 1998]. The motivation behind it was having a dataset of huge images it is sometimes very difficult to train an artificial neural network. The huge numbers of parameters (pixels) not only tend to take a lot of time to train but also are sometimes prone to overfitting.

Convolutional neural networks take the advantage of locally weight sharing within individual feature maps without compromising on the complexity of the model. The overall structure of a typical CNN is quite similar to an artificial neural network except that here the layers are arranged in three dimensions and each neuron is only connected to a small region of the previous layer. The input to a CNN is three-dimensional image (three color channels) and the output can be different classes or continuous values depending on whether it's a classification problem or regression problem.

5.2. Training of Convolutional Neural Network

Various experiments were performed on the data. All the experiments were performed on NVIDIA GTX-650 graphics card with 1GB memory. In all the experiments a validation split of 80/20 (training/testing) was used and data was also shuffled before training. A schematic diagram of our CNN architecture is shown in Figure 5.1.

5.2.1. Experiment 1: Training of a Basic CNN

For this experiment we filtered out samples with missing data, i.e. we used only those training images, which have all the 30 target values. The total number of these images was 2140 (30% of original data). Further all the training data was normalized including the target values.

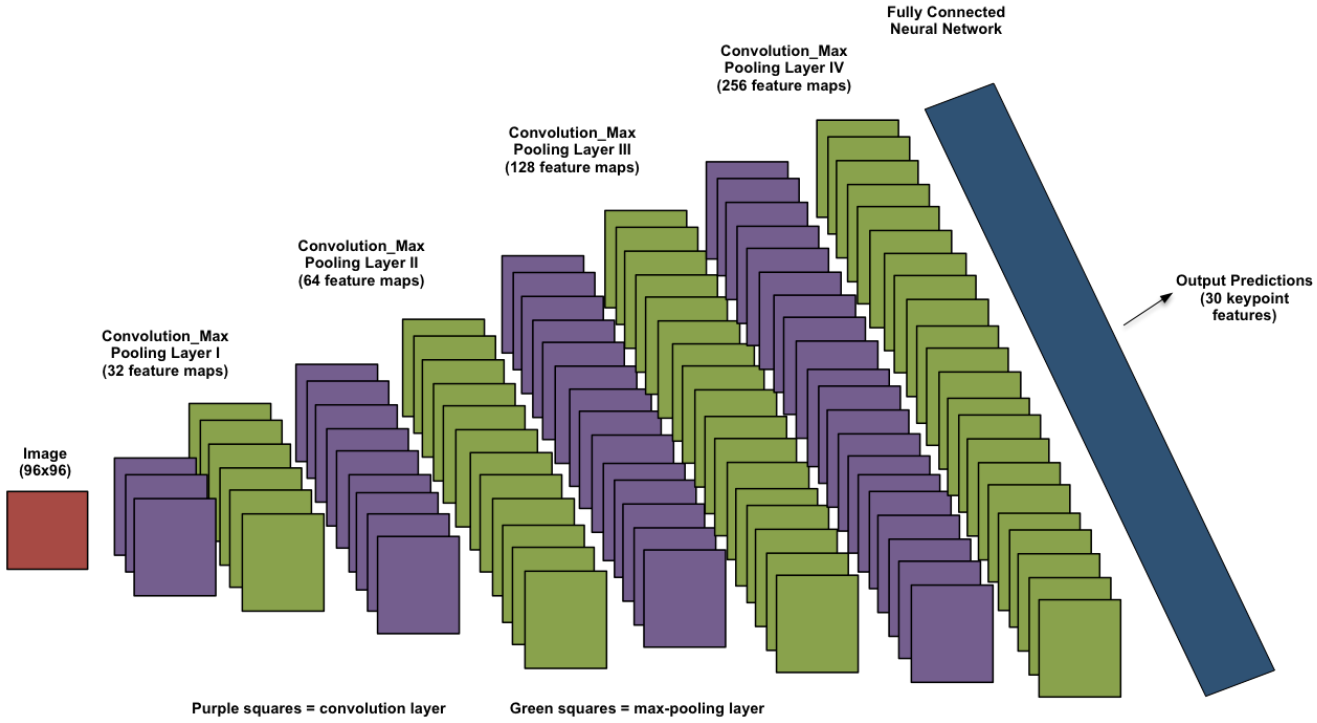

Fig 5.1. Schematic diagram of our CNN architecture

The following model architecture was used in this experiment:

1. Input Image: 96x96x1 (the images are gray scale thus have only one channel)
2. Four Convolution-Max_Pooling layers: (each with no zero padding)

1st Layer: Number of filters/feature maps: 32 Convolution_Filter_size = 3x3, Pooling_filter =2x2 (stride=2)
2nd Layer: Number of filters/feature maps: 64 Convolution_Filter_size = 2x2, Pooling_filter =2x2 (stride=2)
3rd Layer: Number of filters/feature maps: 128 Convolution_Filter_size = 2x2, Pooling_filter =2x2 (stride=2)
4th Layer: Number of filters/feature maps: 256 Convolution_Filter_size = 2x2, Pooling_filter =2x2 (stride=2)

3. One Fully connected layer: Number of neurons/hidden units = 500
4. Final Output Layer: Number of outputs = 30 (since there were 30 keypoint locations)

The activation function for each neuron in the convolutional layers was a RELU non-linearity (max(0,x)) function. Since this is a regression problem, we do not use any output activation function. Hence the output was a linear combination of the activations in the previous fully connected layer. For backpropagation, the objective function that was minimized was the mean squared error (MSE). First we kept the training rate and momentum constant and soon realized that the training process is extremely slow and cannot finish in a reasonable time *(CNN_1)*. Hence to achieve results faster, learning rate was set to decrease linearly while the momentum was linearly increased with the number of epochs *(CNN_2)*. To avoid running into memory issues, as the GPU memory is only 1GB, the batch size was set to 64. Training this CNN model was much faster than before as can been seen in Figure 4.2 when we compare the cyan and the magenta curve.

We stopped training this model after 600 epochs and realized that the gap between training error curve and validation curve slowly started increasing suggesting overfitting. From around 400 epoch in Figure 4.2 one can observe the divergence of solid and magenta curve.

5.2.2. Experiment 2: Training of the previous CNN with data augmentation

To reduce overfitting we next tried training the same CNN with more training data. More training data was obtained by flipping the images horizontally and also flipping corresponding target values. However instead of training for 600 epochs we trained this model *(CNN_3)*, for more than 600 epochs because we thought that since this model has more data to train (keeping all the parameter same as previous) it would take a longer time. However we observed that both training and validation error reduced, but overfitting still persisted and was quite prominent from 700 epoch onwards. This experiment suggests that one needs to have a lot of data to obtain good performance on a CNN.

5.2.3. Experiment 3: Training of the previous CNN with dropout

In order to address the overfitting issue, we next tried a recently introduced technique called dropout. Dropout in convolutional neural networks was recently introduced by [Hinton et al, 2014].The idea behind this technique is that we are dropping (setting to zero) random neurons (activation units) in each convolutional layer during training. For each training image different set of neurons are dropped. This prevents the neurons from co-adapting i.e. it prevents interdependencies from emerging between neurons. This allows the model to learn a more robust relationship, thereby reducing overfitting.

For training, the same CNN architecture was used as in the previous experiments with the addition of these dropout layers, between each pooling layer with varying percentages *(CNN_4)*. To analyze the results, plotting a relationship between training error and validation error like in previous experiments is difficult as the training error reported by the model is with dropout while the validation error is without. Hence to test our model, we ran it on the test set and recovered its Kaggle score. With the model being trained for 1000 epoch the RMSE on test data was 3.45, however when we increased the number of epochs to 3000 the RMSE dropped to 3.21. This was the best performance, which we could achieve with this model as the RMSE didn't improve on further increasing the epochs.

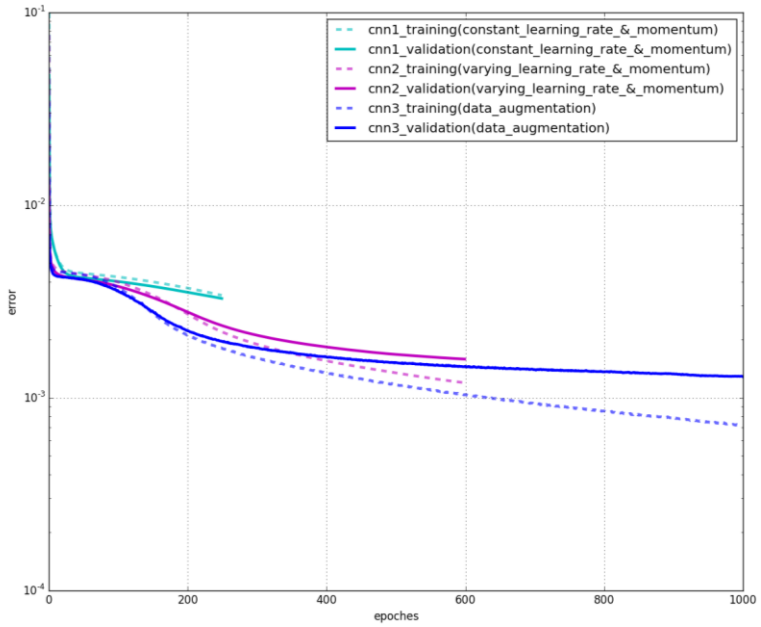

Fig 5.2. Training & Validation error for experiment 1 & 2

5.3. Summary of Results

In this chapter, we explore to apply a convolutional neural network to predict the feature points locations. We further improve its training speed by adjusting the learning rate and momentum. In order to address overfitting, we further investigated some regularization techniques like data augmentation and dropout. The combination of all the above techniques gave the best RMSE result of 3.21 as shown in Table 5.1.

Table 5.1. Results of different CNN models

Model	Model configurations	Score on Kaggle (RMSE)
CNN_2	CNN with varying learning rate and momentum with 600 epochs	3.42401
CNN_3	CNN with data augmentation with 1000 epochs	3.30947
CNN_4	CNN with dropout with 3000 epochs	3.21959

6. Conclusion

In our project, a wide spectrum of learning models are applied to detect facial keypoints and their performance are improved coherently. Linear regression with traditional CV method does not perform well due to its inherent inability in feature selection and complexity control. SVMs obtain much more appealing results partially owing to the kernel flexibility. Yet its power pales in comparison to neural network especially when the models are dealing with high-dimension problem. In the end, CNN shows its exceptional advantage when addressing image-related learning problems.

7. Reference

Chang, Chih-Chung, and Chih-Jen Lin. "LIBSVM: a library for support vector machines." ACM Transactions on Intelligent Systems and Technology (TIST)2.3 (2011): 27.

Dalal, Navneet, and Bill Triggs. "Histograms of oriented gradients for human detection." Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 1. IEEE, 2005.

Efroymson, M. A. "Multiple regression analysis." Mathematical methods for digital computers 1 (1960): 191-203.

Hsu, Chih-Wei, Chih-Chung Chang, and Chih-Jen Lin. "A practical guide to support vector classification." (2003).

LeCun, Yann, et al. "Gradient-based learning applied to document recognition."Proceedings of the IEEE 86.11 (1998): 2278-2324.

Lin, Hsuan-Tien, and Chih-Jen Lin. "A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods." submitted to Neural Computation (2003): 1-32.

Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." The Journal of Machine Learning Research 15.1 (2014): 1929-1958.

Appendix. A Work Assignment

Through the whole project, our group members discuss periodically and achieve good cooperation. Hao paid most his attention to linear regression CV model and SVMs, Jia mainly focused on feed-forward neural network and Nitin primarily concentrated on convolutional neural network.

Appendix. B HOG Implementation

For each 96×96 face, we divide it into 12×12 blocks and each block is a 8×8 matrix. In each block we construct a 9-bin histogram for orientation statistics. Each pixel within the block casts a weighted vote for the orientation-based histogram channel based on the valued found in the gradient computation and the histogram channels are evenly spread over 0 to 180 degree into 9 bins. In short, we bin gradients from 8×8 pixel neighborhoods into 9 orientations. In order to account for noise such as changes in illumination and contrast, we normalize the 9 channels and make them sum to 1.