CSCI 1850 Project Report: DenseEnsemble: Predicting Gene Expression from Histone Modifications

Team Persimmon:
Zachary Hoffman, Mateo Encarnacion
Brown University
7th April 2020

Abstract

Histone modifications play a large part in controlling gene regulation. Our model called DenseEnsemble uses bagging predictors and a state of the art convolutional neural network to evaluate gene expression values from histone modifications. We generate separate versions of our predictor for each cell line and average these values to make predictions over unknown cells. We found that bagging reduced overfitting and provided substantial gains in accuracy on the test set.

1. Introduction

In gene expression prediction, models must make predictions over cells not present in the training data. This causes a large drop in accuracy between the train and test datasets. Highly complex models such as deep neural networks suffer greatly from this problem as they already face issues with overfitting. Ensemble methods have historically found substantial gains in accuracy in cases where the model overfits on training data [1].

While a number of existing papers propose deep learning algorithms for predicting gene expression from histone modifications, ensemble approaches are a relatively untouched area of study. The authors of DeepChrome [5] developed a convolutional model to predict gene expression from histone modification. CNN models such as DeepChrome perform well on many tasks because they provide automatic feature extraction and capture nonlinear relationships. However, with the rise of residual networks and an improvements in computer hardware, the computer vision community has recently developed novel CNN architectures with far more layers than the DeepChrome architecture.

In this paper, we use an ensemble of a highly successful residual network called DenseNet [2] to make accurate gene expression predictions.

2. Method

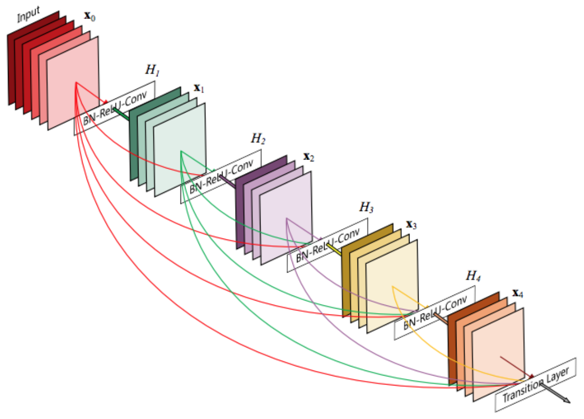

Figure 1. A 5-layer dense block with a growth rate of 4. Each layer takes all preceding feature-maps as input. [2]

Our model consists of a series of dense blocks. In each dense block, every layer connects to every following layer in a feed-forward fashion (see Figure 1). Therefore, if a dense block consists of L layers, the block contains $(L-1)+(L-2)+...+1$ connections. These connections alleviate the vanishing-gradient problem, because each layer gains direct access to the gradients from the loss function through their feature map passed to the final layer. They also substantially reduce the number of parameters, because parameters are reused across multiple layers. Furthermore, the connections strengthen feature propagation throughout the network. In traditional CNNs, each layer reads the state from its preceding layer and writes to the subsequent layer. The kernels make changes the state but also passes on information that needs to be preserved. Residual networks make this information preservation explicit through concatenation operations or additive identity transformations.

Each dense layer in a block performs four consecutive operations: compression, batch normalization, ReLU, and a $3\times$

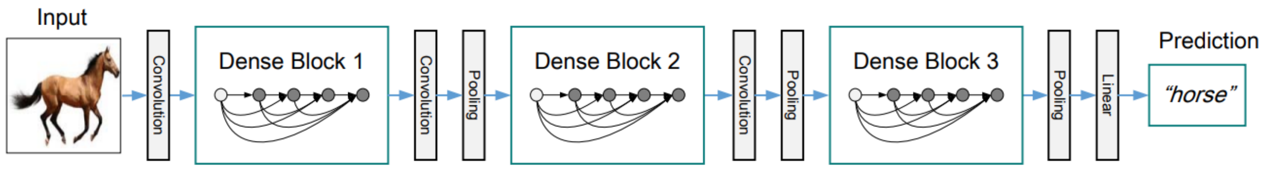

Figure 2. A network three dense blocks. The layers between two adjacent blocks are referred to as transition layers. [2]

3 convolution. The compression operation consists of batch normalization, ReLU, and a 1×1 convolution that acts as a bottleneck layer. Although each layer only produces a fixed number of output feature-maps, they typically have many more inputs (the concatenation of all preceding layer outputs). If each layer produces k feature maps, it follows that the l^{th} layer has $k_0 + k * (l - 1)$ input feature-maps, where k_0 is the number of channels in the input layer. This constant k is referred to as the growth rate of the network. The compression layer uses the 1 x 1 convolution to reduce the number of input feature-maps and improve computational efficiency.

The concatenation operation is not viable when the size of feature-maps changes, so the network is divided into multiple densely connected dense blocks (see Figure 2). A transition layer consisting of a 1 x 1 convolution and pooling layer separates each dense block. To improve compactness, we reduce the number of feature-maps at transition layers. If m is the number of feature maps produces by a dense block. For $0 < \theta < 1$, $\theta * m$ is the amount generated by the transition layer.

We feed the output of the final layer of DenseNet to a fully connected linear layer for the given regression problem.

Our DenseNet model architecture consists of 4 dense blocks each with 3 layers per block. This results in a network with 1 starting convolution, 1 starting pooling layer, 12 densely connected convolutions (4 sets of 3), 12 bottleneck layers (4 sets of 3), 4 transition convolution layers, 4 transition pooling layers, and 1 fully connected linear layer. Thus, our model contains a total of 35 layers. We used a growth rate of 4, a θ of 0.5, and 16 initial feature maps.

We train a separate version of this 35 layer model on each cell line. When predicting the gene expression of cells not represented by the training data, we average the predictions of all models. Otherwise, we use the model trained on the given cell type. We use this ensemble approach, because it greatly reduces overfitting and improves test set accuracy.

3. Experiment Setup

To train and test our model on your own data, please use the command:

```
python histone.py -s -T <path/train.npz>
 -t <path/eval.npz>
```

We train 50 models (1 for each cell line) for 20 epochs (every 152 batches is one epoch) with batch size of 100 and a random $90/10$ training validation split. We use the ADAM optimiser with a learning rate of 0.001.

3.1. Data

The dataset is taken from the class Kaggle competition, and consists of data with the gene expression value and histone mark counts (of five different histone marks) for 56 cell types across 17477 genes. The five histone marks used are H3K27me3, H3K36me3, H3K4me1, H3K4me3, H3K9me. The counts for these marks are measured across 100 neighboring bins of size 100bp; 5 read counts are recorded for each bin, and they are then normalized using the ln(1 + counts) to account for the large range in values. During prepossessing, this data is divided into a training, validation, and test set.

3.2. Training and Testing

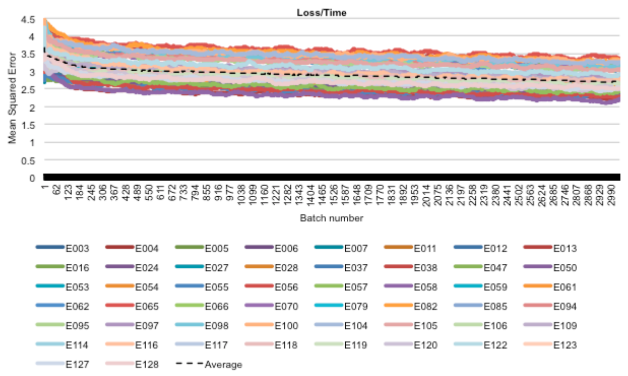

Figure 3. A graph showing the different models' loss values with respect to time.

Figure 3 shows the progression of the training process for all 50 models for the full 20 epochs. Mean squared Error (MSE) is used to calculate the loss value after each batch, and is used as the final evaluation metric for the model. The range of loss values is large, with the worst performing model having a final MSE of 3.33, and the best performing model having a final loss of 2.22 The average MSE at the end of the 20 epochs is 2.72; we decided to end the training here, as at this point the loss begins to stabilize.

4. Results

For our final results on our fully trained DeepEnsemble model, we were able to achieve the following MSE scores: 3.21 on the validation set, 3.40 on the Kaggle public leaderboard test set, and 3.45 on Kaggle private leaderboard test set.

4.1. Baselines

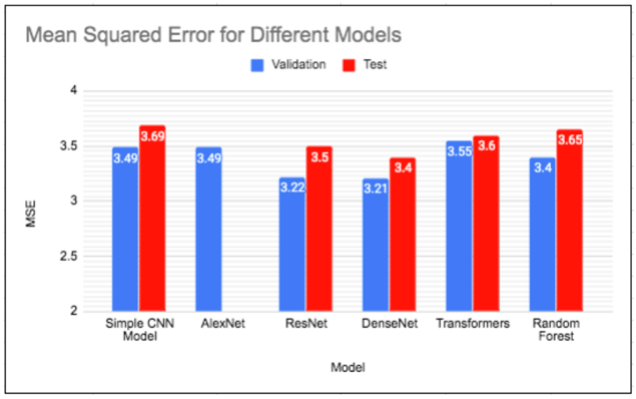

Figure 4. Graph showing performance results.

Along with DenseEnsemble, we experimented with a few different models to gain a baseline and learn about the problem space. Figure 4 shows the performance of these models in comparison to DenseEnsemble. DenseEnsemble performs the best (average MSE of 3.21 on the validation set and an average MSE of 3.40 on the test set), closely followed by ResNet (average MSE of 3.22 on the validation set and an average MSE of 3.50 on the test set). As both architectures use residual connections, it is not surprising that both produce similar MSE values on the validation set. However, DenseEnsemble outperforms ResNet noticeably on the test set, suggesting that DenseEnsemble is less susceptible to over fitting. Simpler CNN architectures were used as well for comparison but had poor results - namely, AlexNet [3], the winner of the LSVRC2012 classification contest, and a simple CNN model (1 convolutional layer, 3 fully connected, with relu and max pooling after the convolutional layer).

Other baselines include Transformers, given their recent success in representing sequences and language models, as well as a Random Forrest model, a different machine learning algorithm based on decision trees.

4.2. Performance Evaluation for different Cell Types

Figure 5 reveals how DenseEnsemble's performance varies with cell type using the MSE values obtained on the validation set. It can be seen that the model for cell E056 performs best with a MSE of 2.51, whereas the model for cell E047 performs the worst with a MSE of 4.25. A MSE

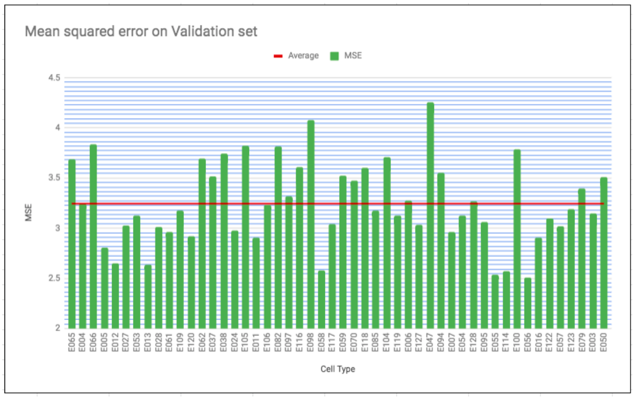

Figure 5. scores for different cell types.

value of 3.24 is the average MSE of all the models. The large range in performance for different cell types may suggest that histone modifications play a larger role in gene expression for certain cell types, but also may be an artifact of the difference in data quality between cell types. In a future study, we would like to investigate why some models work better with different cell types, and how to best fine tune the model for its respective cell type.

4.3. Effects of Histone Marks

Figure 6 shows the activation of different histone marks across the transcription start site (TSS) using a saliency map[4] for six different cell types. The pixels in the map represent the calculated gradient of the predicted gene expression value with respect to the input values. A pixel with a larger gradient signifies that the pixel has a stronger impact on the final output value. The gradient is calculated using PyTorch's autograd package, and has a range of [0.001,0.04]. For a given cell, the lightest pixels represent the input values with the largest gradient (regions of highest activation), whereas darker pixels represent the input values with the smallest gradient (regions of lowest activation).

The different maps shown indicate various interesting observations concerning the relationship between histone marks and gene expression. For one, the different patterns of activation for the six cells, indicate how different regions around the transcription state site (TSS) and different histone marks are more influential on gene expression than others, depending on the cell type. For Cell-type E100 and E012, it is apparent that the most highly activated region when predicting gene expression is between bins 40 and 60, and that H3k27me2 stands out as the most influential histone when predicting gene expression. Other cells such as cell-type E027 show similar trends, but with a less dense and more spread out activation area that is most active between bins 30 and 70. Similarly, H3k27me2 still seems to be the most influential mark for gene expression, but in this case the other histone marks are more evenly activated.

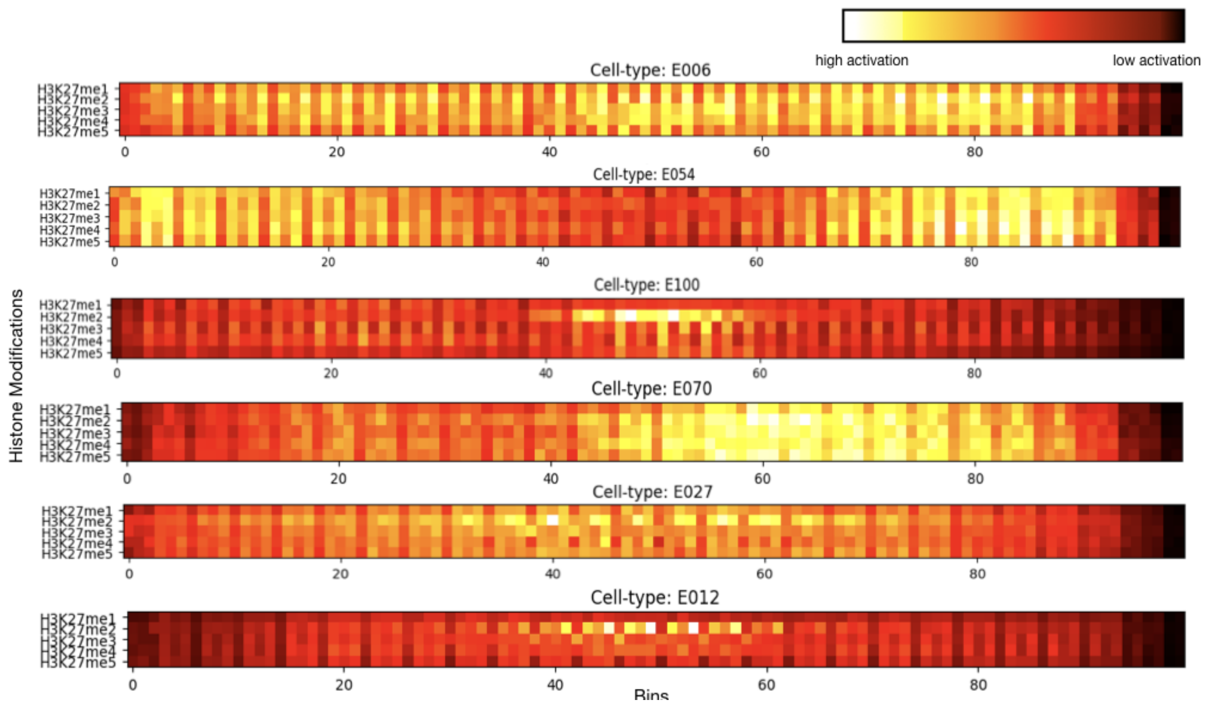

Figure 6. Saliency map for six different cell types

Cell-types E070 and E054 reveal concentrated areas of high activation across all histone marks in specific areas of the TSS - for cell-type E054, these high activation areas are in the start and end of the TSS, while for E070, the high activation area is concentrated towards the end of the TSS (roughly between bins 50 and 90).

Promoter regions bind transcription factors and help initiate transcription. The light areas may correspond to those regions. However, since we don't know the orientation of each of these genes, its hard to draw a conclusive hypothesis.

One common pattern throughout the six maps are the recurring, evenly spaced, low activation (darker) vertical lines across the TSS. We are not fully sure what this suggests, but one answer would be that this reflects how the model divides the different regions along the TSS, and chooses which features are the most important in a very orderly, systematic way, as all the marks seem very methodically placed. Or alternatively, these cells could have some biological property in common which leads to this recurring pattern across the cells. For example, the patterns may be caused by different promoter sequences separated by dead space or repeating copies of gene sequences.

5. Conclusion

In conclusion, DenseEnsemble uses a convolutional neural network based off of DenseNet [2], a highly regarded residual network, with the additional use of bagging as an ensemble method to help prevent overfitting. Our model performed well in comparison to the other baseline methods implemented, and offers insight into how the location of histone marks along the TSS, as well as the specific mark itself affects gene expression. This could help biologists create new experiments and theories for why certain gene mutations and gene expressions appear. In the future, we would like to use different visualization methods such as a CNN filter weights analysis, or a perturbation analysis, to better understand patterns revealing how different histone marks activate certain gene expressions.

References

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996. 1

[2] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. 1, 2, 4

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 3

[4] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014. 3

[5] R. Singh, J. Lanchantin, G. Robins, and Y. Qi. Deepchrome: deep-learning for predicting gene expression from histone modifications. *Bioinformatics*, 32(17):i639–i648, 2016. 1