Deep Learning in Genomics Midterm Project Report

By WakaWaka: Isaac Nathoo and Haider Asif

Introduction:

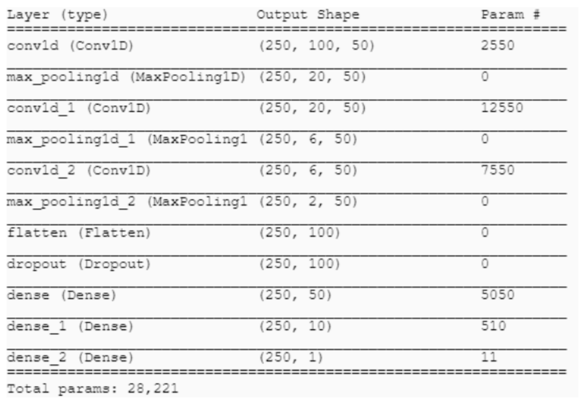
Using histone mark data, our goal was to develop a deep learning model that could predict gene expression in a total of 56 cell types across 17447 genes. The training set contained the histone marks and expression values for 50 cell types and 16000 genes. For each cell type, each gene contained 100 bins which had the histone read counts for 5 histones.

Our final model included three dilated 1D convolutional layers and three fully connected layers. When we trained our model on the entire dataset, we achieved a final training loss (mean squared error) of 3.0212 and a Pearson correlation coefficient of 0.38. Furthermore, we ran k-fold cross-validation for 4 folds using our model and found that the validation loss generally decreased across the epochs. Our final testing mean squared error on the evaluation set with data for 6 held-out cell types across all genes and 1447 held-out genes was 3.43688. Our final testing was fairly large compared to the final training loss, which indicates that our model was overfitting the training data. Although we did try many techniques to reduce the overfitting to the training data, our future work would focus mainly on reducing that, and combining sequential information from the genomic sequences using an LSTM or GRU.

Method:

Our final model architecture was a Keras sequential model defined as follows:

- Conv1D - 50 filters of width 10
- LeakyReLU with alpha=0.05
- MaxPooling1D with pool size of 5
- Conv1D - 50 filters of width 5, dilation rate 3
- LeakyReLU with alpha=0.05
- MaxPooling1D with pool size of 3
- Conv1D - 50 filters of width 3, dilation rate 2
- LeakyReLU with alpha=0.05
- MaxPooling1D with pool size of 3
- Flatten, Dropout with rate 0.3
- Dense Layer with 50 units and LeakyReLU with alpha=0.05
- Dense Layer 10 units and LeakyReLU with alpha=0.05
- Dense Layer with 1 unit, but no activation function



```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (250, 100, 50)            2550
_________________________________________________________________
max_pooling1d (MaxPooling1D) (250, 20, 50)             0
_________________________________________________________________
conv1d_1 (Conv1D)            (250, 20, 50)             12550
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (250, 6, 50)              0
_________________________________________________________________
conv1d_2 (Conv1D)            (250, 6, 50)              7550
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (250, 2, 50)              0
_________________________________________________________________
flatten (Flatten)            (250, 100)                0
_________________________________________________________________
dropout (Dropout)            (250, 100)                0
_________________________________________________________________
dense (Dense)                (250, 50)                 5050
_________________________________________________________________
dense_1 (Dense)              (250, 10)                 510
_________________________________________________________________
dense_2 (Dense)              (250, 1)                  11
=================================================================
Total params: 28,221
```

Figure 1: WakaWaka final model architecture.

The loss function that we used was Mean Squared Error because this was a regression task where we wanted to predict continuous values for gene expression. We also used the Adam optimizer with a learning rate of 0.001 and a batch size of 250, and trained the model for 15 epochs.

We decided to use three 1D convolution layers through biological reasoning about the nature of histone modifications and based on our experimental results, which will be discussed in more detail in the next section. The 1D convolution operation allowed the model to look at all 5 histone marks across a certain number of bins and try to find patterns in the structure. We used multiple convolutional layers so we could extract both local features from the data in earlier convolutions as well as more global features in later convolutions. Furthermore, this number of convolutional layers allowed us to downsample the input data to allow for more efficient learning and to help prevent overfitting, and hopefully to learn more general histone combinations. In addition, we used dilation rates of 3 and 2 in our second and third convolutional layers respectively to capture more of the global signal in the histone data and to possibly capture any longer scale 3D interactions of histone marks across the histone tails. The number of filters for each layer and the width of these filters was determined through k-fold cross validation and model experimentation/hyperparameter tuning.

Next, we chose to use a leaky ReLU activation function because we thought that there might be some model weights that our network learns which might be negative and we wanted to ensure that our model did not set those weights to 0 with normal ReLU activation. Instead, we decided to add a little bit of leak to have small negative weights trickle over while preserving the non-linearity. We also used 1D max pooling layers after each convolution to further assist in downsampling and for the model to focus on the largest weights. Then, we included a dropout layer with a rate of 0.3 to remove some weights from the model and again help prevent overfitting. And finally, we included three fully connected layers for the model to learn how to combine these different weights into a single output value for gene expression.

<u>Experiment Setup:</u>
Before we arrived at our final model, we went through an iterative process of developing and tuning our model. We started with a very simple model with one convolution layer with 8 filters each of width 4, followed by a ReLU activation and a dropout of 0.3 in the middle, and 1 Dense layer with 1 hidden unit with no activation because we wanted to output the expression values instead of the probabilities like DeepChrome[1] did. This model (Experiment #1) did not perform well for us, as shown in Figure 2 below, we achieved a training accuracy of 3.42 on the training data, and a testing accuracy of 3.67. This sparked our experimentation process in which we conducted a grid search for the sizes of the dense layer(s), the filter size, kernel size, and the dilation rates for the convolution layer(s), and the pool size of the max-pool layer(s). We also searched for the most optimal learning rate, activation function, dropout, padding type, batch size, along with trying out regularization methods like batch normalization.

From our initial experiment it was evident to us that our model was not learning as well, so the goal of our experimental approach was to improve its performance (Figure 2), while making sure that it does not overfit by using k-fold cross validation. To accomplish this we tried increasing the number of dense layers to 3, with hidden unit sizes 64, 32, and 1 (Experiment #2) , based on our hypothesis that increasing the number of layers would make the network more complex, and hence enable it to capture the more complex relationships in the data. This turned out to be the correct hypothesis, but we anticipated overfitting and added more dropout layers between the dense layers with a dropout ratio of 0.3. Our results showed we had a lower training loss 3.37, but a still a higher testing loss 3.697, which meant that our model was still overfitting, and hence

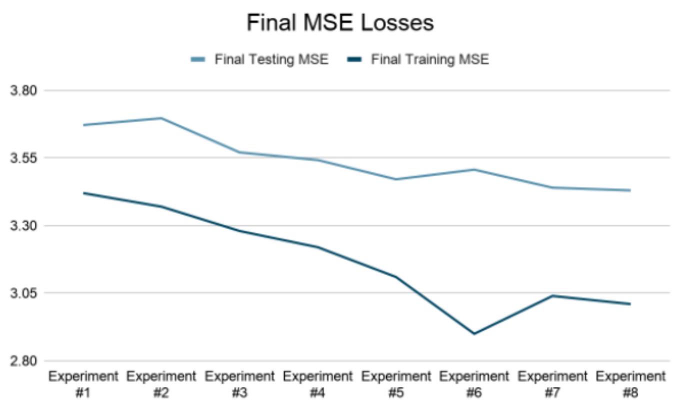


Figure 2: The final training and testing MSE loss across 8 main experiments.

our approach towards reducing it had failed. Our first approach to combat this was to increase the value of the dropout to 0.5, but keep the placement the same, (this is the what is recommended in the Dropout paper that introduced the concept of dropout layers [2]), and to make our model learn further, we increased the sizes of the dense layers to 625, 125, and 1 inspired by DeepChrome [1]. We also changed the activation function from ReLU to LeakyReLU because we hypothesized that there might be negative weights passing through the model, and this was correct as adding a LeakyReLU activation allowed us to slightly capture these negative values. The changes helped the model overfit less and learn better with a training MSE of 3.28, and testing MSE of 3.57 (Experiment #3) .

In an effort to do even better, we hypothesized that dropping 50% of the weights after each dense layer would lose out on too much learning, and would not allow us to learn as much as the model architecture with the other layers was capable of. This is why we moved the dropout layer just after the convolution layer before flattening, and observed that it gave us better training MSE of 3.22 and testing MSE of 3.54 (Experiment #4) . After that, to learn more complex relationships and patterns inside the data, we added 2 more convolutional layers with batch normalization and max-pooling with pool sizes of 3 afterwards (Experiment #5). The rationale behind adding batch normalization was to reduce the noise added due to training the data in small batches and the logic behind adding max-pooling was to capture the most important features in each window. This led to us getting a much lower training loss of 3.11 and testing MSE of 3.47. Following this trend, we added a fourth convolutional layer, increased the dilation rate on the second convolutional layer to 3, and increased the dilation rate on the third convolutional layer to 2 (Experiment #6), which did worse on the testing MSE than previously and caused it to increase to 3.51. Thus, we went back to three convolutional layers but kept the dilation and increased our batch size to 250 (Experiment #7), which improved our performance on the testing data to reach a MSE of 3.44. Next, we tried to decrease the sizes of our dense layers to prevent overfitting, removed batch normalization, lowered the learning rate to 0.001, decreased the dropout to 0.3 and only trained for 15 epochs (Experiment #8). This slightly decreased our testing MSE to 3.43688, which is our best yet, and our training MSE loss to 3.0212.

Results:

After experimentation, the general structure of our model was three convolutional layers with leaky ReLU activation and max pooling, 0.3 dropout, and then three dense layers. We used the Adam optimizer with a learning rate of 0.001 and a batch size of 250, and trained for 15 epochs.

When we ran k-fold cross validation for 4 folds with a 75/25 split in the training data (Figure 3), we found that the training loss was constantly decreasing and the validation loss was also generally decreasing and relatively smooth. The validation loss shows some patterns corresponding to overfitting as it bounces up and down. This suggests that our three convolutional layers may have been learning features that are slightly too specific to the training data. However, another possibility is that there isn't enough information solely within the histone mark data to build a sufficiently robust model. We decreased the size of our model to have less than 30,000 parameters to try to prevent overfitting, but this was the best result we could obtain. Training this model on 100% of the training data resulted in a training MSE loss of 3.0212 and on Kaggle, we achieved a final testing mean squared error (MSE) of 3.43688. Throughout the course of training, the model's loss was consistently decreasing across the epochs (Figure 4). We also chose to cut off the training at 15 epochs because it began plateau afterwards and the loss decreased very slowly. The Pearson correlation coefficient was 0.3821 between the predicted expression values from the training data and the true labels.

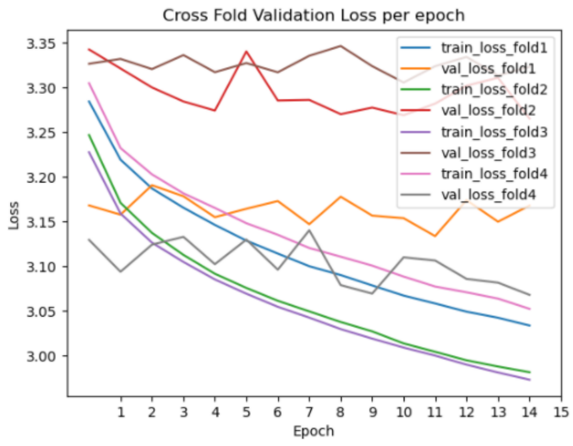


Figure 3: The training and validation losses across 15 epochs using k-fold cross validation with 4 folds.

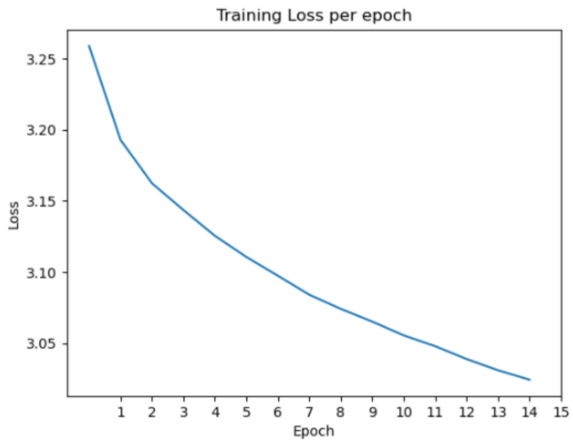


Figure 4: The training loss across 15 epochs for all the training data.

Inspired by some other groups, we tried to incorporate a bidirectional GRU into our model after the convolutional layers and before the dense layers with the hopes that it would learn a sequence of local and global features in the histone mark data that would be important to predicting gene expression values. We tried the bidirectional GRU with 10, 25 and 50 units, but found that they all led to overfitting and worse performance on the testing MSE (above 3.457). Thus, we decided to avoid the GRU approach for now, but this will be something we try again in our future work with the DNA sequence data as we incorporate this into our model.

Some of the innovation in this work was the application of dilated 1D convolutional layers and the use of the leaky ReLU activation function, both of which improved performance. We also demonstrated that adding more convolutional layers and decreasing the size of the dense layers, compared to the DeepChrome model, could improve performance. This is likely because our new model had less parameters and thus overfit less, and because the convolutional layers extracted global features in addition to local ones.

<u>Citations</u>:

1. Singh, R., Lanchantin, J., Robins, G., & Qi, Y. (2016). DeepChrome: deep-learning for predicting gene expression from histone modifications. Bioinformatics, 32(17), i639-i648. [1]
2. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research, 15(1), 1929-1958. [2]