Classifying and Visualizing Splice Junctions with Convolutional Neural Networks

Karun Singh (ks939)

December 13, 2017

1 Introduction

Computational biology has a history of relying on statistical models to make inferences about biological processes. However, these models are not generalizable, since they are often built on a set of assumptions which might not always hold true. Over time, the field has started to adopt more general learning-based approaches to solving classical problems.

These learning-based approaches generally do not depend on any particular underlying model of data. Instead, they learn a representation that is well-suited to the dataset and task at hand. However, there is a common perception that these models are to be treated as magical black-boxes that are somehow, mysteriously learning highly accurate functions.

In this project, I pick a particular class of models – convolutional neural networks (CNNs) – to show that they need not be treated as opaque black-boxes when trained on tasks in the genetics domain, such as pre-mRNA splice junction recognition. Rather than achieving the highest accuracy possible with the CNN on the task, I shift my focus to decoding the implicit function learned by the network. Different methods of doing so were explored, each of which produces unique views of the network's internal knowledge. This learned knowledge also shows a high correlation with pre-existing biological knowledge regarding splice junctions.

2 Background

2.1 Pre-mRNA Splicing

At a high level, the biological process of transcription is responsible for generating functional proteins by decoding DNA. One particular stage of transcription involves converting a pre-mRNA strand to an mRNA (messenger RNA) strand. This conversion manifests itself as a procedure called 'splicing': an editing of the pre-mRNA which results in the removal of certain subsequences called introns. The subsequences that remain, called exons, are joined together to form mRNA, which eventually codes for the generation of a specific protein.

Because of its direct impact on protein generation, splicing is a widely studied process. The way pre-mRNA is spliced explains gene expression and protein diversity, which in turn can impact several other biological processes. Specifically, finding out exactly where on the pre-mRNA strand splicing occurs, and where it does not occur, can yield many insights about the transcription process. The places at which introns are spliced out are called 'splice junctions'. Introns are spliced out precisely because they are flanked by two splice junctions on either end. Further, there are two types of splice junctions – exon-to-intron (EI) junctions, and intron-to-exon (IE) junctions, and they are chemically very different. At the lowest level, splicing is catalyzed by complex molecules called spliceosomes. Thus, the problem of determining if a subsection of the pre-mRNA strand can form a junction or not is at some level equivalent to determining the properties of spliceosome molecules.

In this project, I tackle the splice junction recognition task that can be framed as the following 3-class classification problem: given a window of 60 bases from the pre-mRNA strand, determine whether it is an EI junction, an IE junction, or not a junction at all.

2.2 Convolutional Neural Networks

In the past, statistical models have been used to show the existence of certain well-preserved 'consensus sequences' – that is, sequences whose locations have been found to consistently correlate with the locations of certain types of junctions [BSS00]. However, my objective is to attack this problem through the lens of convolutional neural networks, while showing that their learned function can be decoded and interpreted. If no effort is made to explicitly extract interpretable meaning from neural networks, the perception of them being black-boxes starts to strengthen. On the other hand, being able to concisely explain in words what the network is doing under the hood opens the door for comparisons with other approaches.

The reason for using convolutional neural networks is two-fold:

1. Training deep neural networks does not require us to encode any underlying biological assumptions about the data. The network learns a non-linear function that directly maps the input domain (in our case, sequences of bases) to the output domain (probabilities of being a part of each of the three classes) so as to minimize the classification error.

2. Convolutional neural networks have proven to be highly successful in recent years when applied to image inputs. The reason for this is that they are very well-suited to capturing spatial information in the input, and natural images tend to have plenty of information encoded in local spaces. The input to the splice junction recognition task is not quite an image, but it can be thought of as a row of pixels, each of which represents a base. Within the window of bases, it seems logical to assume that spatiality does encode *some* information; bases that are adjacent to each other should be considered together rather than independently. Throwing away the information about the closeness of bases might be wasteful. With this in mind, I decided to experiment with *convolutional* neural networks. Additionally, convolutional neural networks have very recently been successfully applied to problems within the genetics domain. Some examples of these successes are DeepBind [ADWF15], Deep Motif [LSLQ16] and DeepSplice [ZLML16].

Convolutional neural networks are made up of convolutional filters, which can be thought of as matrices holding certain weights. Each filter convolves its weights with its input to produce an output feature map. The weights of the network are repeatedly updated using variations of regular gradient descent on large batches of data, with the objective of trying to minimize some loss function. Typically, for classification tasks, the network will be created such that it outputs k predictions, one for each of the k classes. These predictions – call them $y_1, y_2, ..., y_k$ – are then passed through a Softmax layer, which normalizes the exponentiated vector y to form a probability vector p. For classification tasks, the loss function used is usually categorical cross-entropy, which encourages the network to increase the output probability of the ground truth class.

3 Method

3.1 Dataset

To carry out the experiments proposed below, I used the Splice-junction Gene Sequences dataset published by UC Irvine [noa]. The dataset contains 3,190 sequences of bases, each of length 60. Each sequence is annotated as belonging to one of three classes:

1. EI (exon-to-intron junction)

2. IE (intron-to-exon junction)

3. N (not-a-junction)

The data is made available as a comma-separated file, where each row represents one example. For all of the evaluations, the data is first randomly shuffled, and then split into a training set and validation set in a 90-10 ratio.

To represent the sequence data, an orthogonal encoding approach is used. Each base in the sequence can be represented using four dimensions. These dimensions correspond to Adenine (A), Cytosine (C), Guanine (G) and Thymine (T) respectively. For example, if an A occurs in the sequence, we represent it using the vector [1,0,0,0], while a G would be represented as [0,0,1,0]. The data is not entirely complete – many bases are partially uncertain. In these cases, the unit is distributed among the candidate bases' dimensions equally. For example, a D in the input sequence represents one of A, G or T, so we represent it as [1/3, 0, 1/3, 1/3].

Note that this orthogonal representation ensures that we are not implicitly encoding anything about correlations among the four bases. Some previous approaches, like [ZLML16], use numbers along the same dimension to encode different bases (i.e., A = 0, C = 1, G = 2, ...). However, this inherently tells the model that there is a stronger correlation in the input space between A's and C's, than there is between A's and G's, which is not true.

3.2 Network architecture

The architecture of the convolutional neural network used is shown in Figure 1. It consists of two layers of 1-dimensional convolutions of length 3, each followed by 1-dimensional max-pooling of length 2, and a ReLU operator. The first layer learns 8 filters, and the second layer learns 16 filters. Finally, the output of the second layer of convolutions is fed into a fully-connected layer with 3 outputs, which are subsequently passed through the Softmax function.

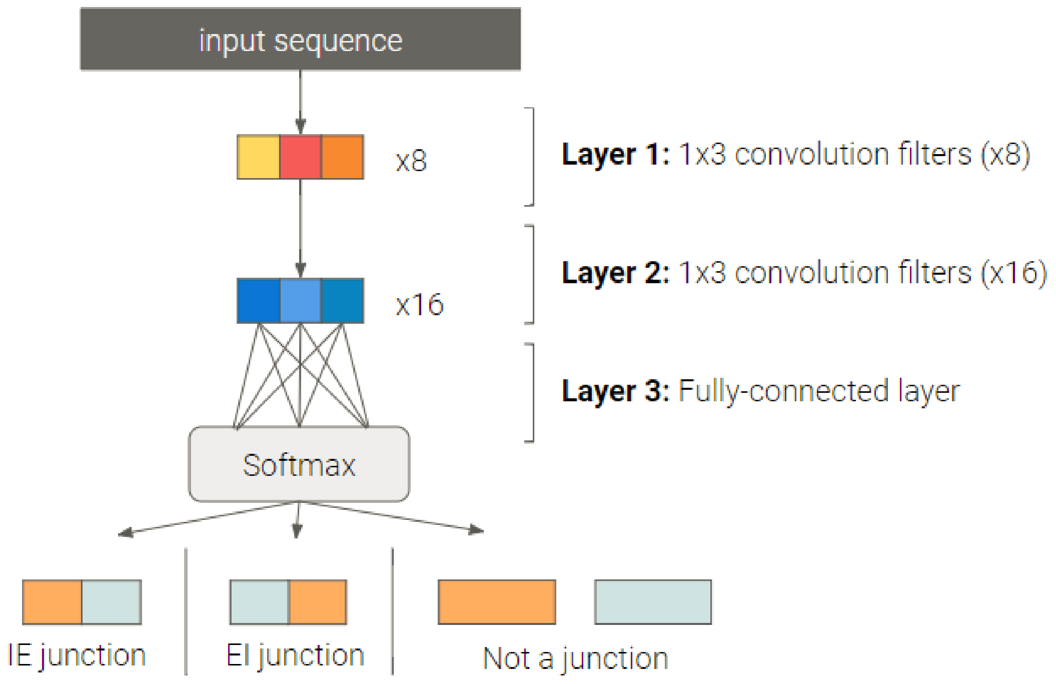
input sequence
x8
Layer 1: 1x3 convolution filters (x8)
x16
Layer 2: 1x3 convolution filters (x16)
Layer 3: Fully-connected layer
Softmax
IE junction
EI junction
Not a junction

Figure 1: Pictorial representation of the network architecture. The output is a probability vector with three elements, corresponding to the three output classes.

3.3 Training process

To construct and train the network, I use the Keras framework [Co15] using a TensorFlow backend. The loss function that we look to minimize is the categorical cross-entropy loss. I use the Adam optimizer [KB14] with default parameters to minimize this loss. The batch size is set to 32, and the training process is run for 30 epochs of the dataset. At each epoch, the network is evaluated on the held-out validation set.

3.4 Visualizations

Once training has completed, the network's learned weights are frozen and used to generate the visualizations. This process is described in more detail in the later sections.

4 Results

4.1 Can the network classify junctions?

The classification problem does not seem to be too hard for the network to learn – it has sufficient capacity to capture almost all the cases presented in the dataset. After just 30 epochs of training, which cumulatively takes under a minute to run, it learns to classify upwards of 93% of the validation examples correctly. Thus, it is successful in learning to detect and classify junctions. A visualization of the validation accuracy tracked over the training epochs is shown in Figure 2

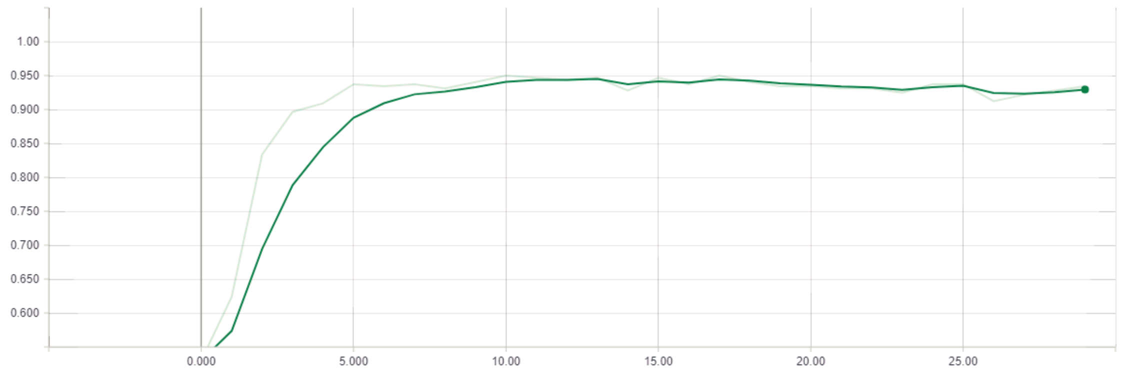

Figure 2: Validation accuracy (y-axis) vs training epochs (x-axis). The accuracy converges to a value between 93% and 94%.

Achieving higher accuracy is not the main goal of this project; rather, the focus is to try and describe what the network has learned in an interpretable manner. With this in mind, the following sections shift focus to the problem of interpreting the network's learned weights.

4.2 How does the network classify junctions?

To decode how exactly the network is classifying windows of bases, I decided to use an approach similar to the class model visualization technique proposed in [SVZ13]. To visualize a certain class, we first freeze the network's weights, and then find the input that maximizes the output probability of that particular class. This is framed as an optimization problem: we start with an input sequence where each base is represented as [0.25, 0.25, 0.25, 0.25], and then use gradient ascent to repeatedly move this 60-base input sequence in the direction of higher activations for the class in question. If the input sequence is called s, α is a step-size (set to 1 in my experiments), and $g_{c,s}$ is the gradient of class c's final probability with respect to the sequence s, then each iteration makes the following update:

$$s = s + \alpha g_{c,s}$$

Eventually, s becomes such that it causes class c's probability to approach 1.0, at which point we have extracted what the network thinks class c should ideally look like. The results on the three classes – EI, IE and N – are shown in Figure 3.

4.3 How do these sequences compare to consensus sequences?

The visualizations in Figure 3 are especially interesting when compared with statistically-derived consensus sequences for EI and IE junctions [BSS00].

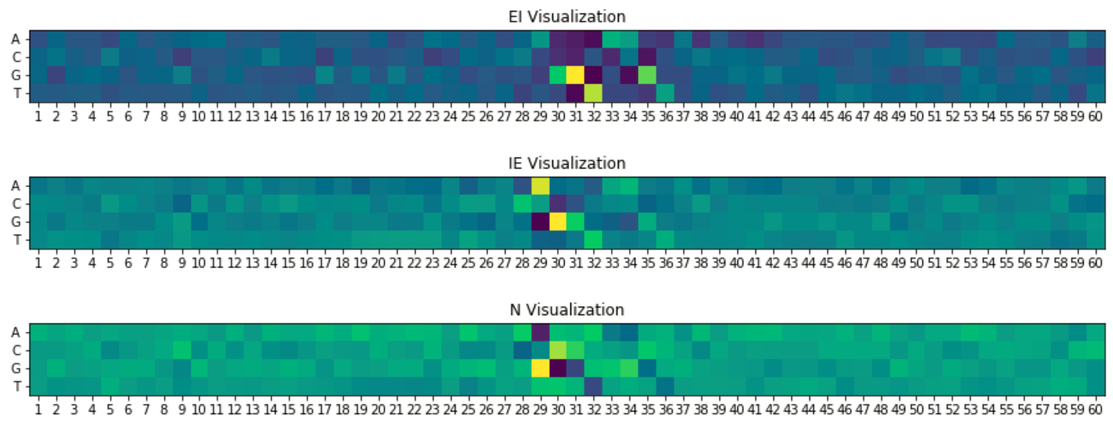

Figure 3: The input sequences that maximize the activations of each of the three classes. From top to bottom, exon-to-intron junction, intron-to-exon junction, not-a-junction.

EI junctions, also called donor splice sites, are known to be marked by GT right after the splicing point. This corresponds directly with the convolutional network's learned representation! It, too, happens to rely on the occurrence of GT at positions 31 and 32 (immediately following the splicing point), while also showing some reliance on the existence of a G at position 30.

A similar trend is observed for IE junctions, which are also called acceptor splice sites. They are known to be marked by the presence of an AG immediately before the ending of the intron. The IE visualization shows the the convolutional network also learns to look for an A at position 29 and a G at position 30, which directly precede the splicing point.

It is also interesting to observe what the network looks for while classifying something as not-a-junction. For starters, it looks for the absence of A and G at positions 29 and 30, marked by the dark black pixels. It also looks for the absence of G and T at positions 31 and 32. Thus, it seems to classify sequences as not-junctions by a process of elimination. Interestingly, the only highly positive signal in the visualization corresponds to the existence of a G at position 29 – a trend that might be worth further exploration.

4.4 Which parts of the sequence are most important to the prediction?

Another way of visualizing the network's learned internal representation is through the use of saliency maps [SVZ13]. Saliency maps look to explain the network's prediction of any one given example by highlighting the features of the input that were most important to the eventual output. More concretely, we first pick a sequence, x, to inspect, and propagate it through till the end of the network. Let the network's predicted class of x be c. Then, to construct a saliency map, we simply extract the gradient $g_{c,x}$: the gradient of the class c's final probability with respect to the input sequence x. The magnitude of this gradient is plotted across all 60 bases to construct a saliency map. The magnitude of the gradient is higher for those bases in the input for which a small change would cause the largest change in prediction. Thus, the highlighted bases were the most 'important' in helping the network make its prediction. Some interesting results are shown in Figures 4, 5, 6, and 7.

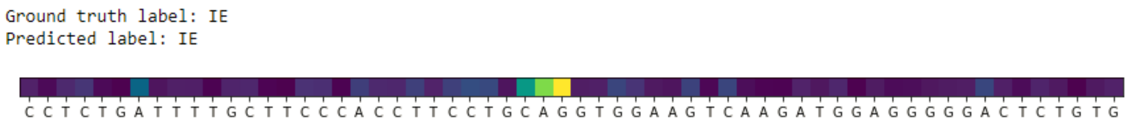
```
Ground truth label: IE
Predicted label: IE
```

Figure 4: Saliency map for an intron-to-exon junction that the network predicts correctly. As the map shows, the network picks up on the AG consensus sequence preceding the splicing point to correctly predict an IE juntion.

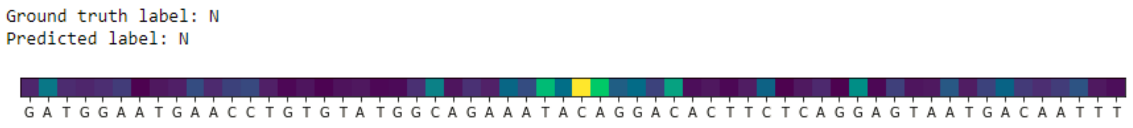
```
Ground truth label: N
Predicted label: N
```

Figure 5: Saliency map for a not-a-junction example that the network predicts correctly. Interestingly, the most influential base for the network's prediction was the C that occurs right after the splicing point. This is in line with the earlier observation that the network classifies not-a-junction examples via a process of elimination. It is not so much the presence of a C, as much as the absence of a G that causes this prediction.

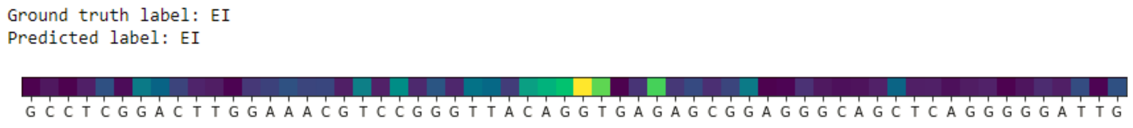
```
Ground truth label: EI
Predicted label: EI
```

Figure 6: Saliency map for an exon-to-intron junction that the network predicts correctly. This example is especially interesting because the sequence contains the consensus sequences at the right places for both types of junctions. However, the saliency map shows that the network deems the GT to be a stronger signal than the AG, hence predicting an EI junction instead of an IE junction.

5 Conclusion

These experiments, although restricted to the task of splice junction detection, are encouraging in that one is able to extract a lot of semantic information from them. Using class visualizations and saliency maps, in most cases one can express in words the reason why the convolutional network predicted what it predicted.

I believe that making learning-based approaches more interpretable is a necessary step if they are to be adopted in application domains that have conventionally relied on more manual techniques. Without interpretable representations, potential beneficiaries of such methods start to think of machine learning models as opaque black-boxes whose internal workings cannot be understood. Thus, I hope that I can continue working on making such models more transparent, especially in the field of genetics and genomics where learning-based approaches can have a giant impact.

6 Source Code

The source code used to implement the methods described above is publicly available on GitHub. Instructions to run the code are in the README file within the repository!

References

[ADWF15] Babak Alipanahi, Andrew Delong, Matthew T. Weirauch, and Brendan J. Frey. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nature Biotechnology*, 33(8):831, August 2015.

[BSS00] M. Burset, I. A. Seledtsov, and V. V. Solovyev. Analysis of canonical and non-canonical splice sites in mammalian genomes. *Nucleic Acids Research*, 28(21):4364–4375, November 2000.

[Co15] François Chollet and others. *Keras*. GitHub, 2015.

[KB14] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. arXiv: 1412.6980.

[LSLQ16] Jack Lanchantin, Ritambhara Singh, Zeming Lin, and Yanjun Qi. Deep Motif: Visualizing Genomic Sequence Classifications. *arXiv:1605.01133 [cs]*, May 2016. arXiv: 1605.01133.

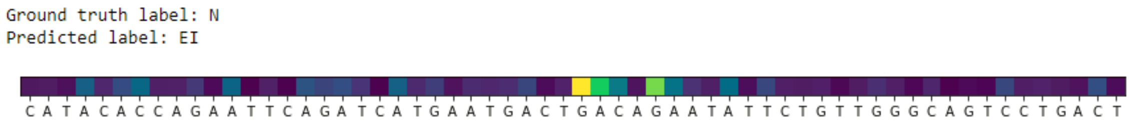
```
Ground truth label: N
Predicted label: EI
```

Figure 7: Saliency map for a not-a-junction example that the network wrongly predicts as an exon-to-intron junction. The saliency map indicates that the G at position 31 was highly influential in the decision-making process. Assessing the class visualization maps from before, it seems as though the network treats a G at position 31 as too strong a signal in favor of EI junctions.

[noa] UCI Machine Learning Repository: Molecular Biology (Splice-junction Gene Sequences) Data Set.

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [cs]*, December 2013. arXiv: 1312.6034.

[ZLML16] Yi Zhang, Xinan Liu, J. N. MacLeod, and Jinze Liu. DeepSplice: Deep classification of novel splice junctions revealed by RNA-seq. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 330–333, December 2016.