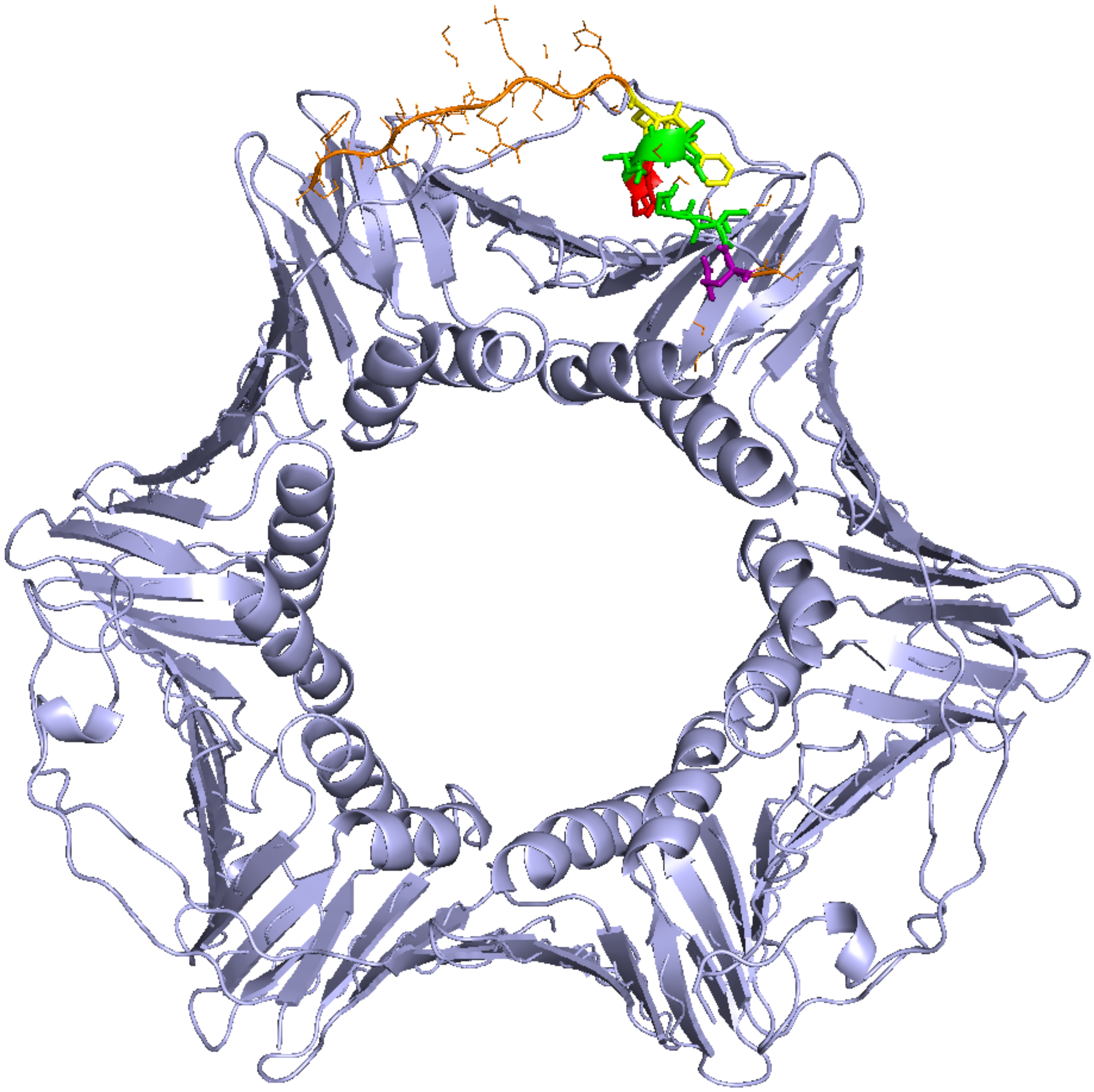
Feature-learning in Short Linear Motifs (SLiMs) and Characterizing SLiMs in their Structural Context

Wouter Krogh Boomsma

William Paul Bullock

Thomas Wim Hamelryck



Abstract

Short linear motifs (SLiMs) perform numerous functions, acting to arbitrate protein-protein interactions despite being intrinsically disordered and having only a few conserved residues. Since their discovery there has been increasing interest in their study for potential medical benefit.

In this study a general model for capturing the structural environment that SLiMs use to bind to their target was created, so as to identify the conserved 'hotspots' essential to the SLiMs operation, and to potentially suggest alternate conformations for that SLiM.

It is possible to visualize the atomic structure of an amino acid residue as an image on a 3D spherical grid. This process produces a 3D array of values representing an amino acid residue.

From here image recognition techniques commonly used in machine learning such as convolution-neural networks were applied to make inferences about the class of amino acid present at a position, based on observing the 3D image of the residue that position, and having undergone supervised learning.

This has allowed the creation of a model, which when sufficiently well trained; can seemingly successfully identify the majority of conserved positions in a SLiM, from having been given its atomistic structure, but definitely requiring extensive further testing and with room for improvement due to with an poor to mediocre accuracy.

Nonetheless the model and concept both have good potential to be of use to computational biology, and with further alterations and perhaps expansion on the core concept (e.g. including torsion angles to infer neighbouring structures too), they may prove more promising still.

Introduction

Short linear motifs (SLiMs) are short stretches of protein structure, (of varying levels of conservation) averaging 6 amino acids long, with only a of these few residues (usually roughly 1/3 of the length of the SliM) being used in their processes. SLiMs are Often separate from other functional properties of the molecule in which they occur. SLiMs are generally intrinsically disordered and lack a well-defined tertiary structure in their native state but they adapt to a template provided by their structured counterpart upon binding. Nonetheless, they perform numerous functions, acting to arbitrate protein-protein interactions. This somewhat contradicts the orthodox idea of proteins requiring a well-defined tertiary structure in order to apply their function. (Davey, Cyert and Moses, 2015)

Since their discovery SLiMs, have been found to be of importance to a diverse abundance of functions. Including, but not limited to: assisting in protein complex assembly, recruiting substrates to modifying enzymes, control protein stability, proteolytic cleavage and structural modification. (Davey, Cyert and Moses, 2015)

Perhaps unsurprisingly there has been an increase in the interest of research into SLiMs' roles in disease and potential uses as novel drug-targets. (Davey, Cyert and Moses, 2015; Wells and McClendon, 2007). Concurrently interest in having available computational models for the recognition, manipulation and recognition of SLiMs is also growing.

To further the knowledge of this field, a general model for capturing the structural environment that SLiMs use to bind to their target was created, thus allowing the user to understand which residues are most likely the conserved 'hotspots' essential for binding in each SLiM . As well as allowing the user to predict alternative sequences for each SLiM of interest. Thus visualizing other SLiMs potentially capable of fitting the same binding pocket.

It is possible to visualize the atomic structure of an amino acid residues as an image on a 3D spherical grid. This process produces a 3D array of values representing an amino acid residue.

After this had been achieved, it was possible to apply machine learning techniques commonly used to solve problems in image recognition to train a model that recognises residues by their atomic structure. This model can then be given the atomic structure of an unknown residue, and it will output a prediction the most likely amino acid labels it has learned for that residue. The model utilized in this study was a convolutional neural network architecture, commonly used for 3D image analysis in other fields, tailored to purpose.

The model was trained on a homology reduced dataset of 2336 proteins from Protein Data Bank (PDB), iterating through their amino acid residues in mini-batches, for each accepting the features of that residue's atomic structure as a 3D array of values. (Berman *et al.*, 2000)

The model was then tested on known two known SLiM conformations commonly found in many of the interaction partners of proliferating cell nuclear antigen (PCNA).

PCNA is a cellular hub protein central to an intricate protein network that mediates DNA replication and repair, epigenetics and chromatin remodelling. Many of the interaction partners of PCNA possess and intrinsically disordered region, within which is a short amino acid sequence called a PIP-box. The PIP-box is typically 8 residues long with the motif QxxΨxxϑϑ where Ψ is an aliphatic hydrophobic residue and ϑ is an aromatic residue, Q is specifically glutamate, and x can be any amino acid. This can also be written under the notation Qxx[L/V/M]xx[F/Y][F/Y]. (Moldovan, Pfander and Jentsch, 2007; Stoimenov and Helleday, 2009; Marabini *et al.*, 2017).

Of particular interest in this study are the crystal structures of p21 bound to human PCNA (Gulbis *et al.*, 1996). As well as; Human PCNA variant complexed with p21. (Duffy, Hilbert and Kelch, 2016). henceforth referred to as 1AXC and 5E0U respectively.

Both of these PCNA ligands contain the PIP-box motif expressed as the 8 residues QTSMTDFY (gln-Thr-Ser-Met-Thr-Asp-Phe-Tyr) within a disordered region.

After the model was been trained, it was tested by being sequentially presented with the atomistic features of the amino acid resides in the PIP-box from these two structures to see how closely it could predict the known conformation, as well as observing which other residues scored highly in each position, so that we might infer potential alternate conformations.

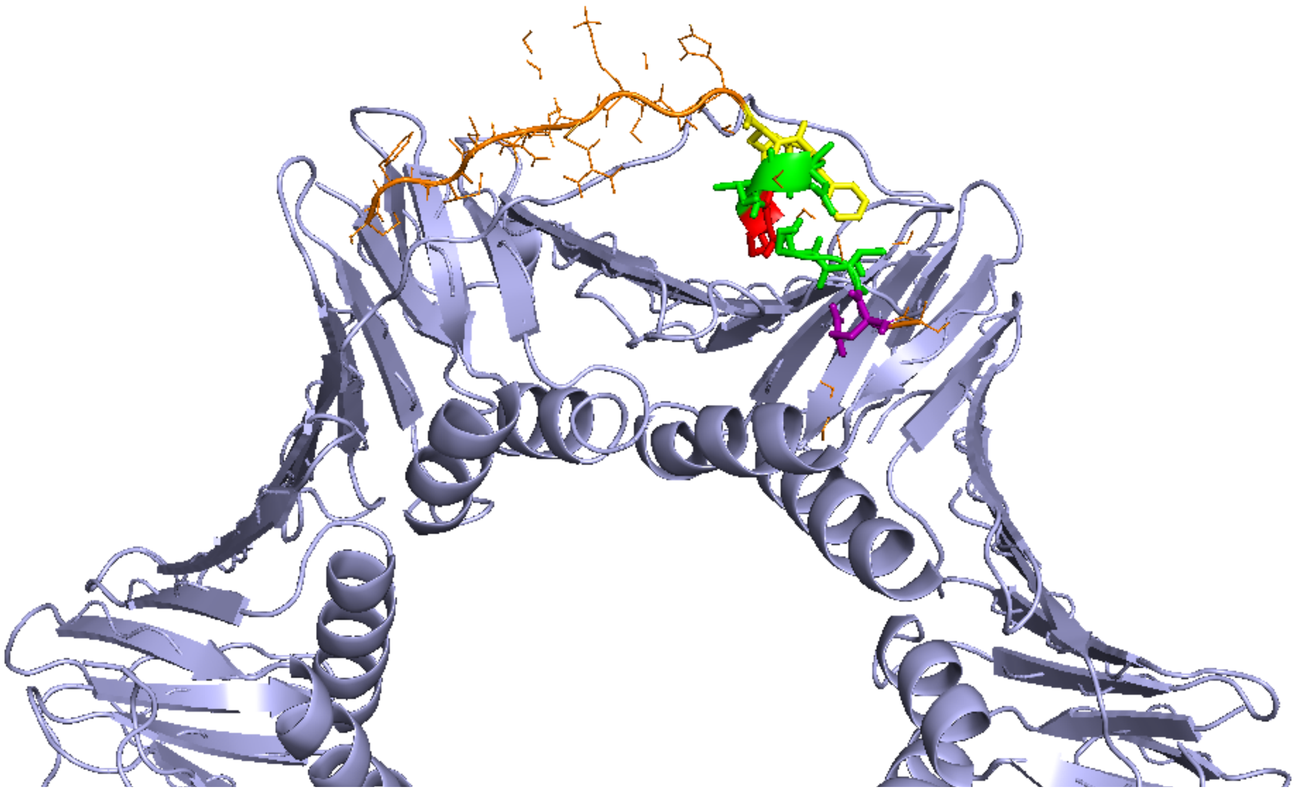


Figure 1. Above. 1AXC; PCNA complexed with a 22 residue peptide derived from P21. The coloured areas indicate the entirety of the P21 disordered region. The glutamate is coloured purple, the variable regions are green, the aliphatic hydrophobic residue is red, the aromatic residues are yellow. The orange regions are disordered sequence belonging to P21 that does not comprise the PIP-box SLiM.

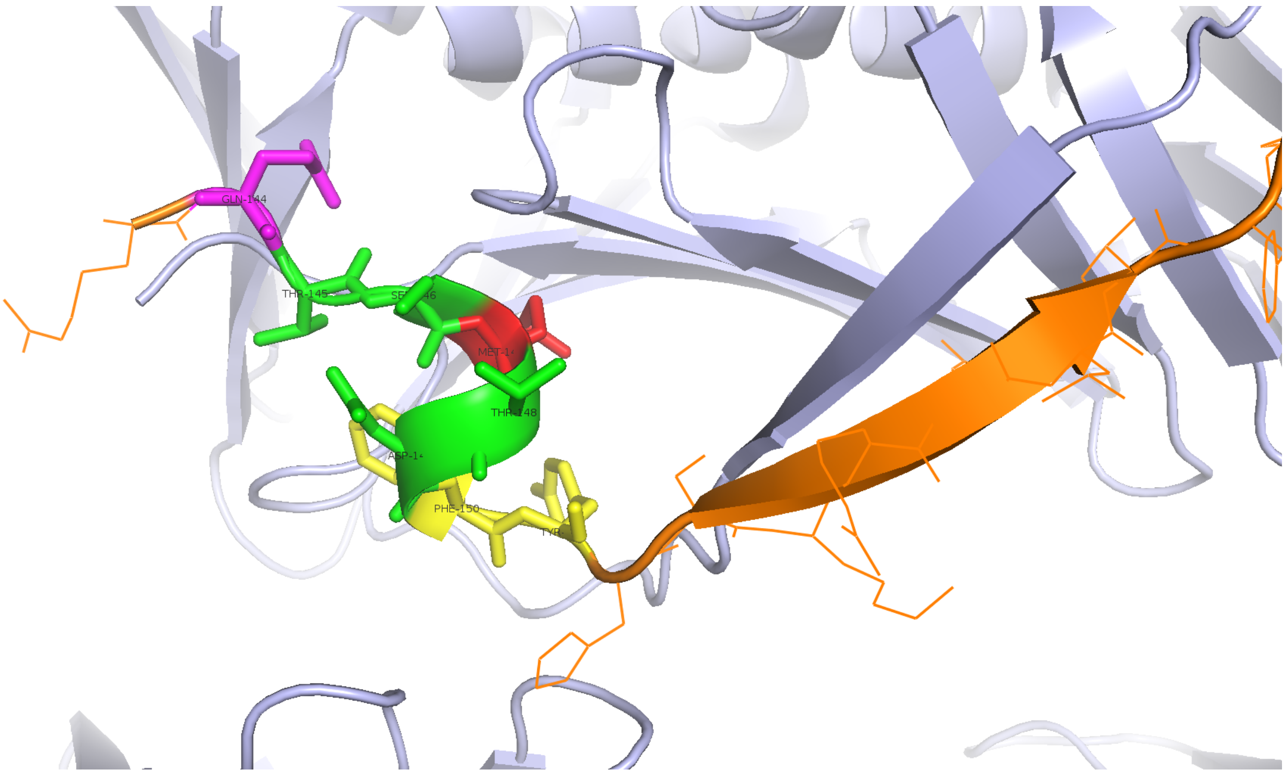


Figure 2. Above. 5E0U; Close up of a diseased PCNA complexed with a 22 residue peptide derived from P21. The coloured areas indicate the entirety of the P21 disordered region. The glutamate is coloured purple, the variable regions are green, the aliphatic hydrophobic residue is red, the aromatic residues are yellow. The orange regions are the disordered sequence belonging to P21 that does not comprise the PIP-box SLiM.

Materials

Software

- PyMOL for biological structure imaging.
- The python programming language was used, for the development of all machine learning models, particularly using the module TensorFlow.
- R statistical programming language was used for data analysis, ggseqplot was used to generate sequence logo graphs. (Wagih, 2017)

PDB structures for model training and testing

A homology reduced dataset of 2336 proteins and their atomic features from PDB (Berman *et al.*, 2000) supplied by Wouter Krogh Boomsma.

Deepfold_grid & Deepfold_batch_factory

Two python modules integral to this study; deepfold_grid is used to apply the atomistic features of amino acids to 3d spherical grids, deepfold_batch_factory is used to partition the aforementioned PDB database into training and validation sets, and call them into python in batches. *both written by Wouter Boomsma & Jes Frellsen. (Boomsma and Frellsen, 2017a, 2017b)*

Operating systems

All implementations of the models discussed in this study were run on the servers provided at hecaton.binf.ku.dk by the University of Copenhagen Bioinformatics Centre.

Methods

Amino Acid residues as 3D data

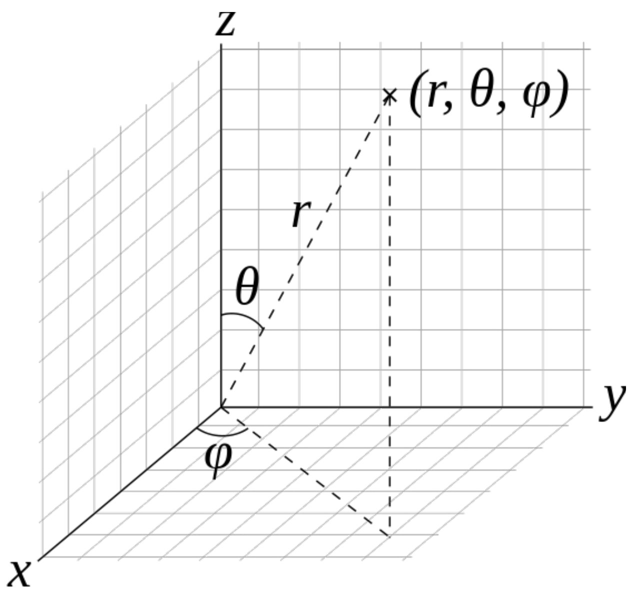


Figure 3. Above. Diagram of the constituent mathematical components of a spherical grid (physics notation used). Image from (Dmcq, 2012)

Firstly it is necessary to be able to present the 3D atomic structure of an amino acid residue in a way which can be interpreted by the model.

The atomistic features of many protein structures are readily available from the PDB as numpy data files (.npz). From here a python script (Boomsma and Frellsen, 2017b) was used to map the the the 3D structure of the residue to a spherical grid, taking the C-alpha atom as a zenith.

The radial distance r is of 0-12Å fit into 0.5Å bins, the polar angle θ 0-180 degrees and Azimuthal angle φ 360 degrees are fit into squares of bin width ~2.36 and 2.4 respectively. Using this system a processed amino acid structure outputs a [24 x 76 x 151] spherical grid with all the constituent atoms of the amino acid mapped to points on that grid. Atoms are not permitted to overlap. Figure 3 shows a representation of the three components involved in forming a spherical grid.

Convolutional Neural networks

Now that we are capable of representing each amino acid residue as a 3D image, identifying amino acid residues in a SLiM and predicting alternative conformations becomes an image recognition and classification problem. Convolutional neural networks (CNNs) are deep machine learning methods commonly used to solve such problems, such a method was also implemented here.

Convolutional neural networks work by accepting an array of values (often representative of an image), performing multiple convolutions to draw information about parameters that determine the prominent features of the data, increasing the data's depth. The data's size is often downscaled, but new depth retained so as to further accentuate the main features and eliminate subtler inferences. Then the processed data is fed through a multi-layer perceptron in order to classify the data into one or more predetermined categories.

The general architecture of a simple 2D CNN is as follows:

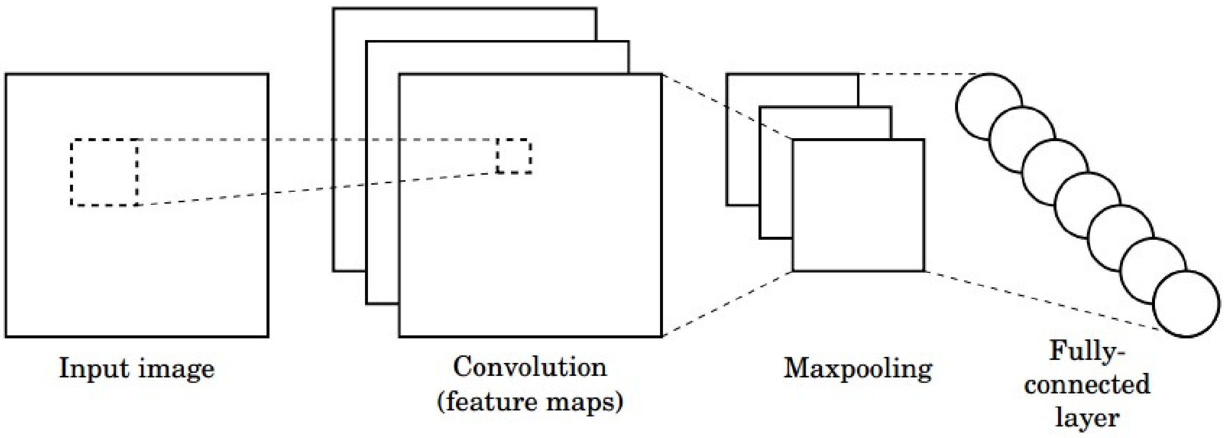


Figure 4. Above. Diagram of the fundamentals of a convolutional neural network. Image from (Pavlovsky, 2017).

There are four main operations in any CNN as illustrated in figure 4:

- Convolution
- Non Linearity (Commonly ReLU)
- Spatial pooling (max-pooling)
- Classification (Fully Connected Layer)

1. Convolution

The image, represented as an array of values, is input to the model, kernels (sometimes called filters) of a pre-determined size and amount are then applied over the the array. The size of the kernels, and their step size (whether the kernels will overlap, or leave gaps in the initial matrix, and how much by) will determine the size of the output matrix, the feature map.

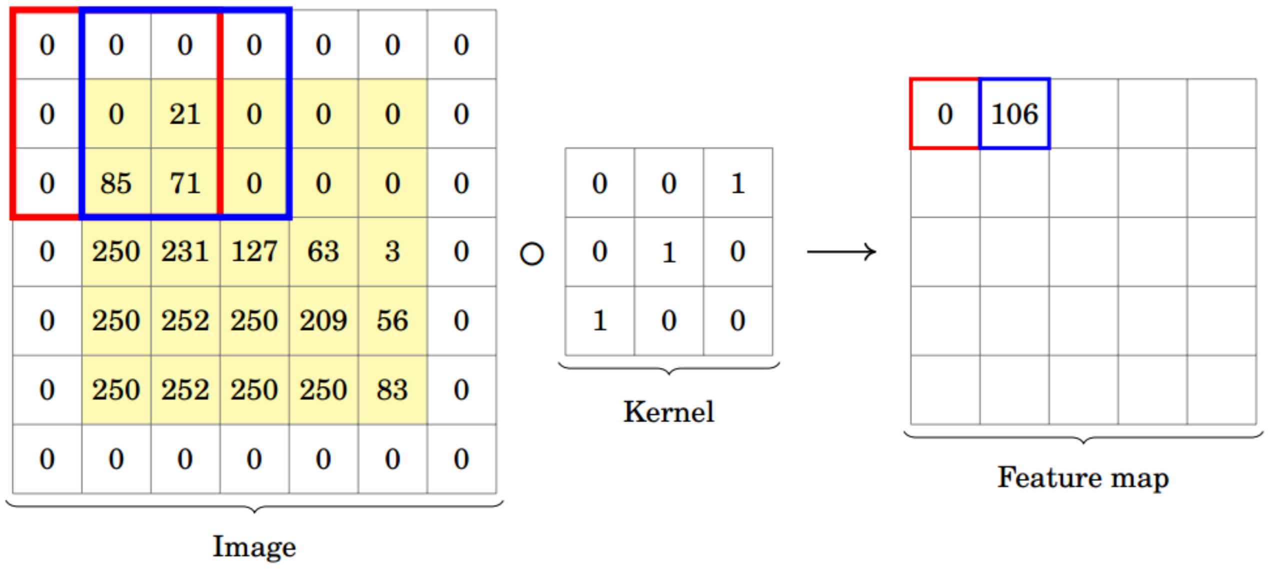


Figure 5. Above. Diagram of the matrix multiplication of an input image and a kernel to transform the input image into a feature map. Image from (Pavlovsky, 2017).

For example, as seen in figure 5, if one has a 7x7 array of values, and applies a 3x3 filter over the array with a step size of one, a feature map of size 5x5 will be created.

The original input image is painted yellow and the 0s in white are added artificially, this is commonly referred to as padding, and is optional.

Without padding one would be able to convolve only 3 times and the feature map would be only 3x3 pixels large. This can potentially be a problem if one intends to perform further convolutions on the resultant feature map.

Kernels containing different values (weights of the convolution), will produce feature maps of varying values from the same input, by using multiple kernels with varying internal values, multiple feature maps will be created, each capturing different features from the input data (often referred to as depth). To complete each feature map, a bias offset is added element wise.

The values within the kernels and the bias offset are typically not defined by the user, they are learned by the model during the training process. Other parameters, such as kernel size, step size, number of kernels, padding and general model architecture are all determined by the user however.

Mathematically, in this instance, convolution is simply the mathematical operation on two different functions to give a third. In this case, if all three arrays were to be expressed as a functions; the feature map is the integral of the point-wise multiplication of the kernel and the input array and is the amount that input array is translated.

After the feature map is complete, a non-linear activation function is applied before it passes to the next stage.

2. Non-linearity

f(x)=max(0,x)

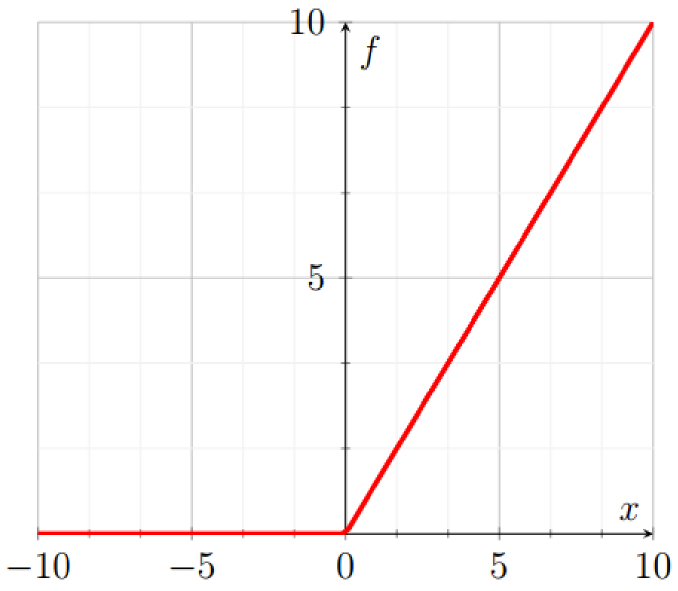


Figure 6. Above. Equation and graph for the Rectified Linear Unit non-linear operation function. Image from (Vojtech Pavlovsky, 2017).

After every convolution a non-linear operation is applied element wise to each of the feature maps. This is done to ensure that no value is ever less than 0. There are various methods of doing this, such as the tanh or sigmoid functions. A commonly employed simple method is the Rectified Linear Unit non-linear operation (ReLU) and is the method used in this experiment.

Practically it is computationally faster than its alternatives, which require more complex mathematical operations.

3. Spatial pooling

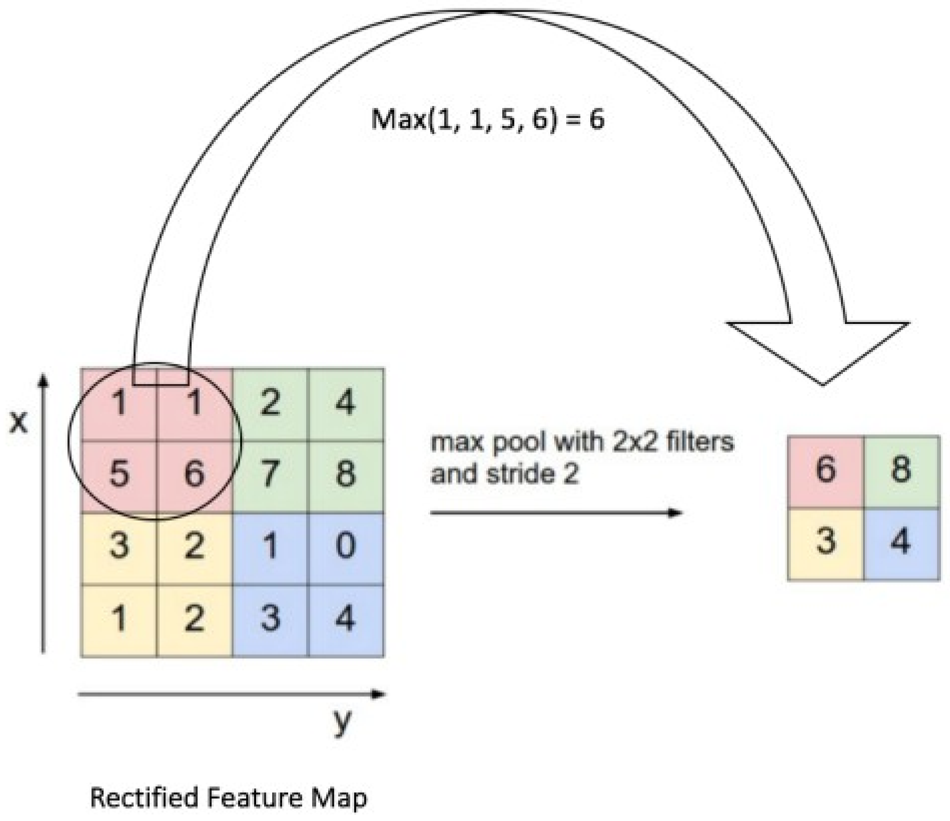


Figure 7. Above. Diagram illustrating downscaling using max-pooling. Image from (Ujjwalkarn, 2016)

As the number of feature maps increase, caused by using an increased number of kernels, It can become important to reduce the dimensionality of each feature map so as to make it computationally manageable. This is typically achieved by one of various methods of spatial pooling.

In the case of max-pooling one defines a window of the feature map and takes only the largest value from that window to represent the larger area of the feature map. As shown in the figure 7.

Sum-pooling or average pooling are also available options, though typically max-pooling seems the most effective, and is the only form of pooling used in the experiment conducted here.

4. Classification (fully connected layer a.k.a. densely connected laye)

Finally this downscaled, deeper image is used as an input for a fully connected layer (FCL), which is commonly a multi-layer perceptron with one or more hidden layers.

The fully connected layer is a neural network of its own right, specifically a multi-layer perceptron. Many simple deep learning neural networks contain only this layer. For the data handled in this experiment, it is not appropriate to feed the input data immediately into this layer however. This is primarily because we are concerned with the spatial recognition of the features of our image; rather than just recognising patterns in the input data, we want to infer meaning from the euclidean distance of the data-points. Furthermore this approach is much more operationally manageable, as it is capable of learning key features of the dataset, whilst downscaling the initial input; this is more suitable for the huge amounts of input data we will present to the model. For other data sets, different neural network models maybe more appropriate.

Each node in every layer of the perceptron is fully connected to every node of the layer before it, hence the nomenclature of 'fully connected layer'.

The FCL performs the classification of the initial input image. It accepts a flattened 1D vector form of the feature maps we have generated (so a 32 x 32 array of depth 16 will give a vector of 16384 values) as an inputs (X_i) and processes them through at least one hidden layer (a_i) , which then passes to an output, the predicted classification of the input image ($\hat{y}_k$).

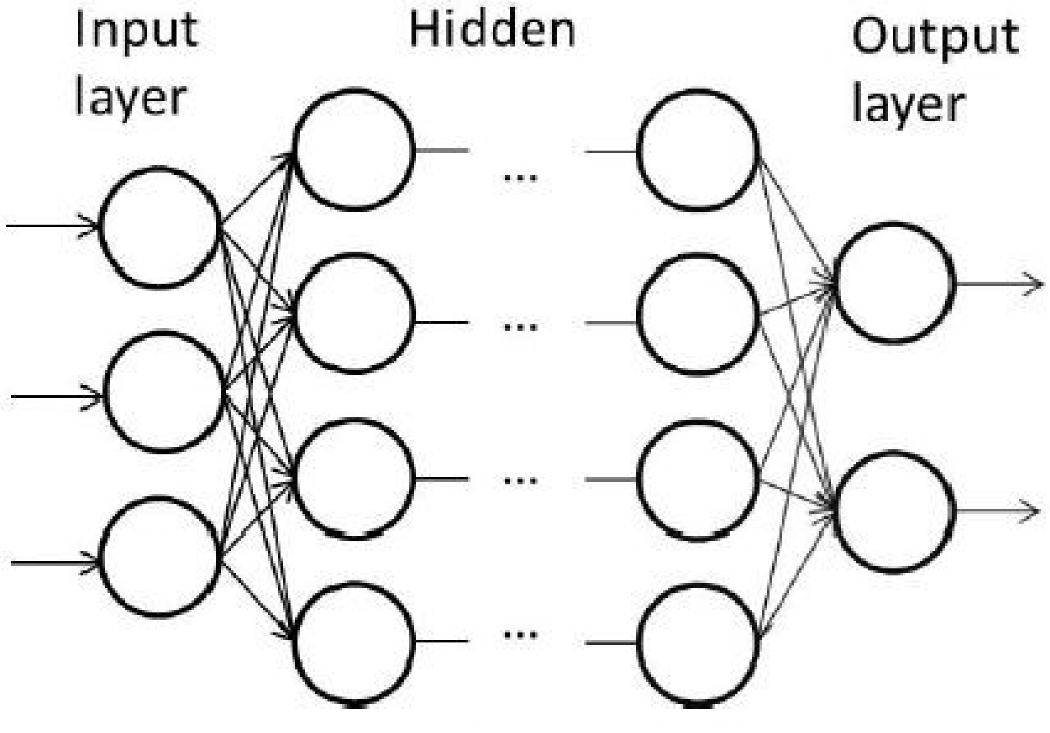


Figure 8. Above. Diagram of a typical multi-layer perceptron. Image from (Ponce et al., 2014)

Figure 8 shows a generic template for a multi-layer perceptron, and figure 9 shows the operations occurring at each node interaction.

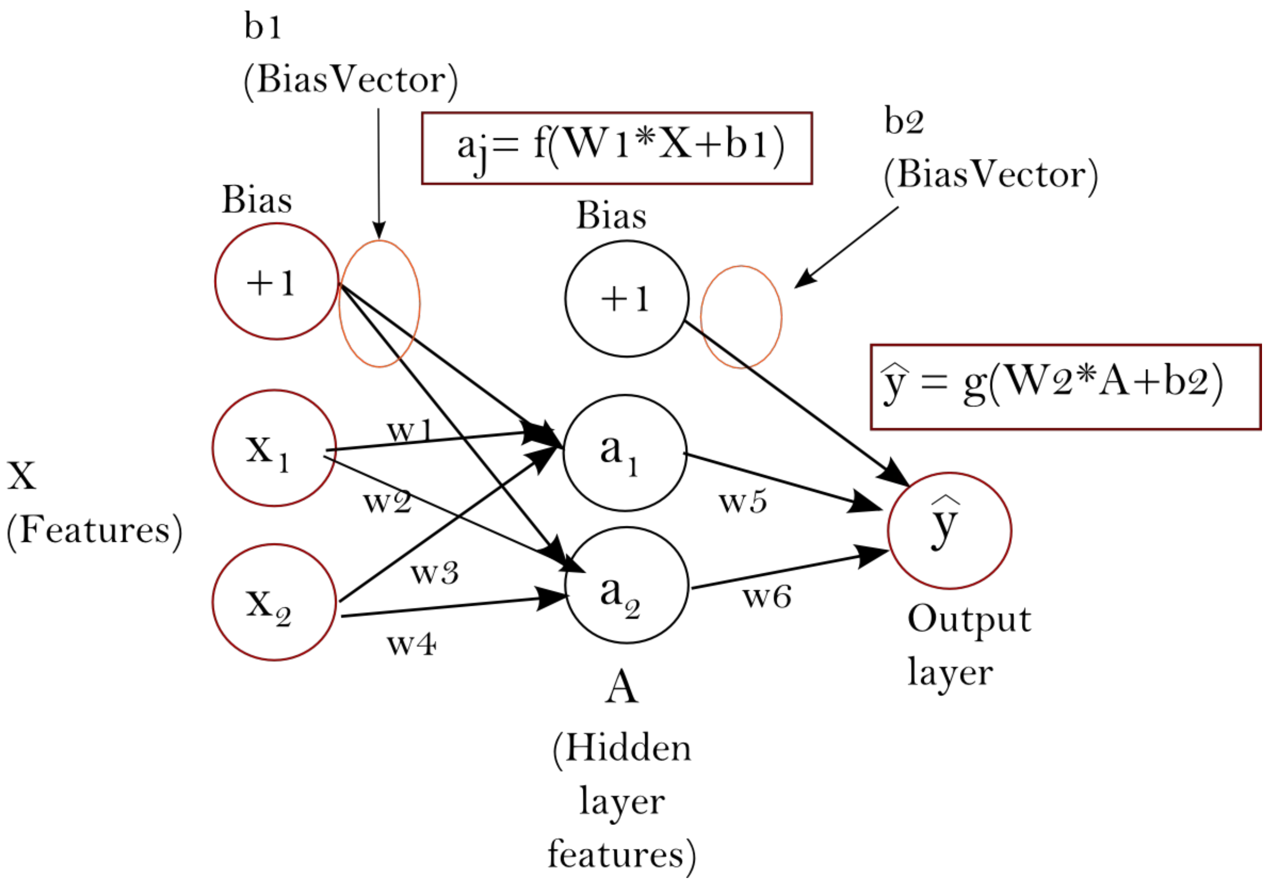


Figure 9. *Above. Diagram of the operations involved in the forward propagation of a typical multi-layer perceptron.*

A multi-layer perceptron makes predictions by forward propagation; a series of matrix multiplications and activation functions.

Each value is multiplied by a weight as it progresses forward into the hidden layer, and then has a bias offset added to it, the weights are unique to each pathway, the bias is common to each layer. The node then has a non-linear operation is applied as an activation function. once again this is commonly ReLU.

The output from this node then passes to other nodes in another hidden layer, where the process is repeated with another set of weights and a bias, or to an output layer.

At the output layer, the final weights and final bias offset are applied and a different activation function is used. This is to cause the output to be a distribution of probabilities.

The softmax function is commonly used in order to achieve this, in simple terms it takes a vector of n arbitrary values, and compresses them to a vector of n values in the range of 0-1, where $\sum n = 1$.

Mathematically, the softmax function is simply the exponential of each element in the prediction score vector divided by the exponential of the sum of the exponential of the total as shown in figure 10:

$$\mathrm{softmax}(y)_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

Figure 10. Above. The Softmax equation

For example; an input of 7 classes with values [1, 2, 3, 4, 1, 2, 3], becomes [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175] after softmax is applied.

The output has most of its weight where the '4' was in the original input. The softmax function acts to highlight the values with the most weight and suppress the signal from values which are significantly below this maximum.

In this example, the model now has the likelihood of the initial input image belonging to each of the seven classes; here it can be seen the 4th class is the most likely classification.

Finally, one now has a working model. however we need to train this model to be able so the weights and biases used in the convolutions and in the FCL are representative of the data we are trying to interpret.

Training the model

Training the model means learning the optimal values for parameters used (the weights and biases of the convolutional layer and the FCL). Initial values are chosen randomly, and then are optimized via the process described in this section.

The output form softmax allows for the mathematical comparison of the similarity of two outputs; one can exploit this to define the distance between the class predictions the model produces ($\hat{y}$), and the true value of the class (y), this is commonly achieved using a loss function.

After this distance has been quantified (known as the loss/error/cost) via forward propagation, an optimization function and a large data set of known outcomes can be used to attempt to minimize this value to make the model more accurate. This is known as supervised learning.

The cross-entropy loss is one example of a loss function, though there are many other alternatives. Cross-entropy loss in essence simply sums over all training samples and adds to the loss if we predicted the incorrect class, the greater the distance between y and $\hat{y}$, the greater the loss.

Mathematically this can be defined as in figure 11:

$$E(y, \hat{y}) = \sum_{i=1} y_i log(\hat{y}_i)$$

Figure 11. Above. Cross-entropy loss function.

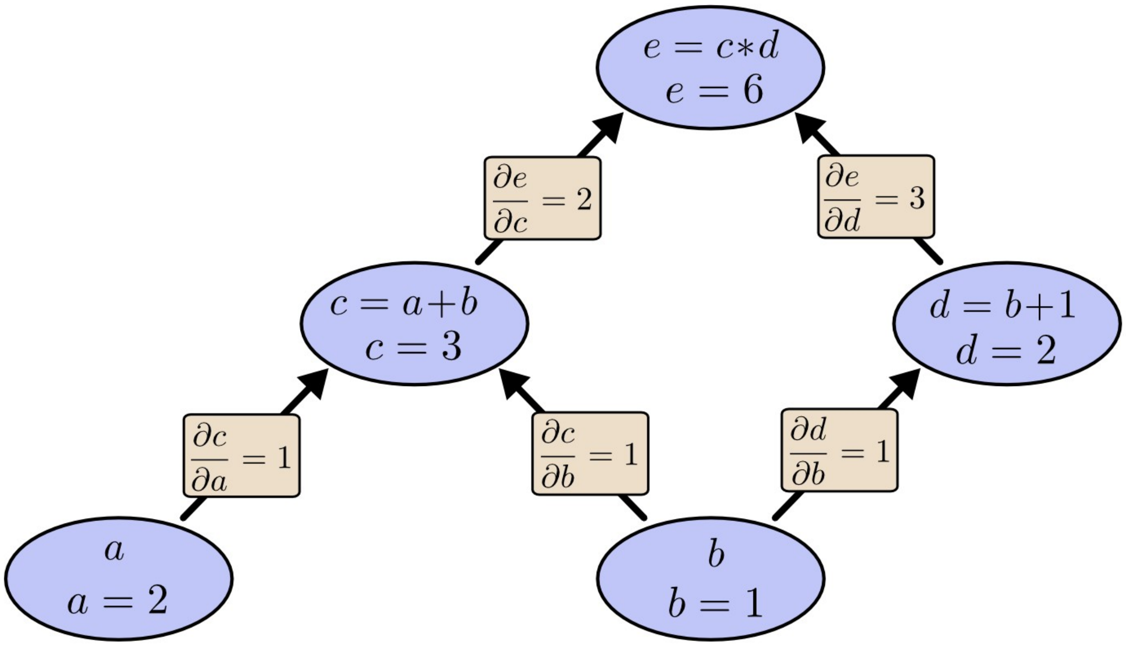
Where y is the true output value, ŷ is the prediction of y, for i training samples.

The overall goal in the process of training is to find the values for existing parameters that minimize the loss function, to find the values that allow us to make predictions closest to the true outcome. In order to find the minimum of the loss function; an optimization function is used.

A common choice of optimisation function is gradient descent, which iteratively moves along a function to find the local minimum of that function. This is achieved by taking steps proportional to the negative of the gradient of the loss function at the current point (on a graph: moving down the slope towards the minimum value).

The contribution of each node towards the total error is calculated by iterating backwards from the output nodes, and the weights corresponding to each node are altered slightly in order to minimise the error that individual node is producing. Thus the models parameters are improved to fit the training data. These two intertwined steps are referred to as back-propagation algorithm and weight update.

Mathematically the steps comprising the back-propagation algorithm rely on calculating the partial derivative of the error with respect to each weight, this informs us how much a change in that weight (and the direction of change) will effect the loss function. This is achieved via utilizing the chain rule multiple times. And is illustrated in Figure 12.



$e = c*d$
$e = 6$
$\frac{\partial e}{\partial c} = 2$
$\frac{\partial e}{\partial d} = 3$
$c = a+b$
$c = 3$
$d = b+1$
$d = 2$
$\frac{\partial c}{\partial a} = 1$
$\frac{\partial c}{\partial b} = 1$
$\frac{\partial d}{\partial b} = 1$
a
$a = 2$
b
$b = 1$

Figure 12. Above. Diagram representing back-propagation Image from (Colah, 2015)

In Figure 12, the nodes A & B contribute to C, B contributes to C & D, and in the next layer; C &D contribute to the output node E.

From forward propagation the value of the output and how each node contributed to that output is known. Because he error of total output is also known; the contribution towards that error that the closest nodes made can be calculated. Iterating further backwards, layer by layer allows the error of all nodes to be quantified this way. Using the chain rule in this way its possible to also find the partial derivative of each node with respect to the output or the node above it.

Once we have this partial derivative, we have the gradient of the loss function with respect to that weight, at that point, and can update the weight (make a step) towards minimising that loss.

The weight update is computed by taking a step in the opposite direction of the cost gradient, which can be defined as:

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j},$$

Figure 13. Above. Typical weight update function.

Where, for j weights, η the learning rate, J is the loss function, and δw is the partial derivative of the error with respect to that weight . The change in weight, which is added to the old weight, is equal to the product of the learning rate and the gradient, multiplied by -1.

The back-propagation and weight update steps are iterated until until some threshold for the number of iterations has been reached, or when the next step can only act so as to increase the loss function (the gradient is at (or very near) a minimum).

Generally, back-propagation is no longer done manually, many advanced tool-kits are available to automate this process such as the TensorFlow module for the programming language Python, which was used in this study.

Often these processes can by computationally demanding, and it is implausible to update the weights of the model for every single training example computed at once in one iteration. Especially in instances where their maybe tens of thousands of training examples. Iterating through each of the training examples individually can take a vast amount of time. As such training the model is commonly run in smaller batches. A go-to method is mini-batch gradient descent, and is the method used in this study.

In mini-bath gradient descent, the training data is split into small batches, (the size of which is decided by the user) then these are used to and update the model together as one. Implementations can either use the average or sum of the loss for the batch then make adjustments to the weights so that the value and the variance of the loss is minimized. This is less robust then taking each individual training example into account in it’s own right, but is greatly less demanding in time.

After all batches of the training data have been iterated though (perhaps multiple passes are performed over the whole training data set (sometimes called epochs)). The weights will be adjusted to allow the model to make accurate predictions of the classes on outside data as far as the model architecture and quality of the training data allows.

Testing the model

In order to validate the accuracy of our model, we present data with known outcomes, that the model has not seen yet (test data) to the model and allow it to make predictions, but don't allow it to know the true values, or train off of the loss.

We then simply compare these predictions with the true values and aggregate how often the the model predicted correctly into an accuracy score.

This gives an indication of how trustworthy results will be when predictions made on future data with unknown outcomes will be.

It is important to realise that inadequate training data will never allow the user to produce a model useful in implementation, and this is of paramount importance to any machine learning model. With good training data; the tweaking of the hyper-parameters of a model (learning rate, batch size, number of layers, number of filters) are the determining factors of accuracy. Optimizing these can be an arduous involving trial and error. The complexity of models can vary greatly depending on the type of data being analysed.

More complex models

The model as described above is a very simple convolutional neural-network, more complex models may have several layers of convolutions, spatial pooling, and FCLs.

Furthermore the number of kernels in each convolution and the padding used, the size of the downscaling in the spatial pooling and the number of hidden layers and nodes in each FCL, and the learning rate in training can be varied by the user for their data. Figure 14 below shows a blueprint of a much more intricate and computationally demanding convolutional neural-network model utilized in a research paper.

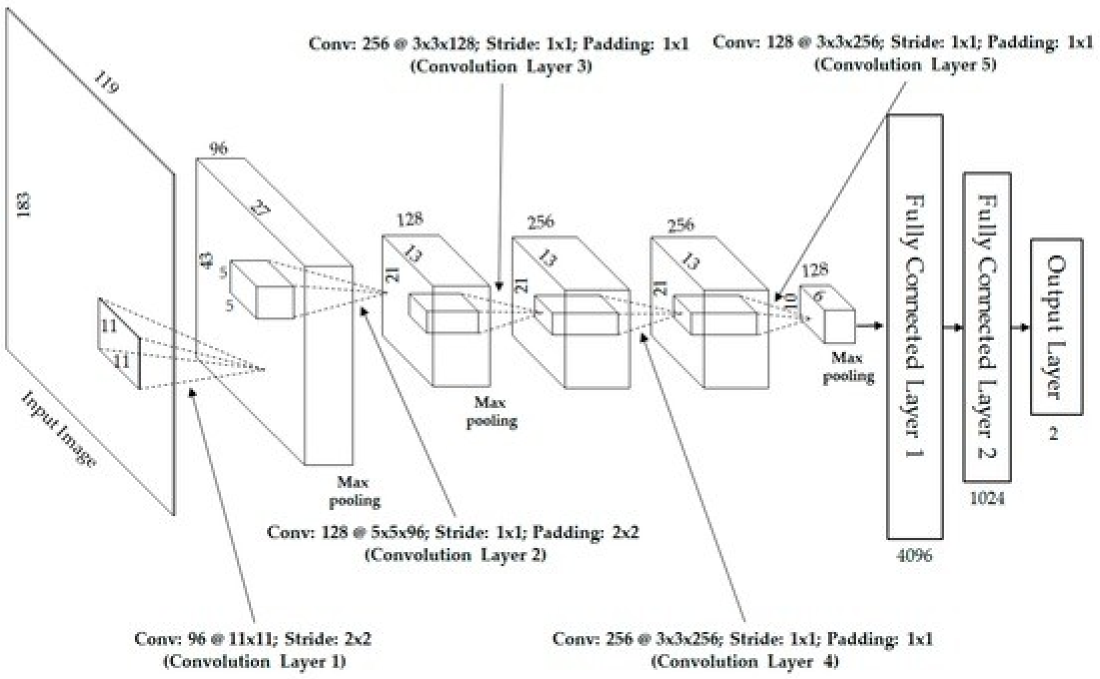


Figure 14. *Above. Diagram of a complex and well-developed convolutional neural-network architecture. Image from (Nguyen et al., 2017)*

Program architecture used in this study

The architecture of the model created for this study has some deviations from the general model, so as to tailor it for the data utilized in this study.

One of the primary differences lies in that the the input image to the convolutional neural-network was 3D not 2D. This meant that the kernels and feature maps were also in 3d, as well as the max-pooling. For both the convolutions and max-pooling padding was used so as to preserve image size.

The shape of the input data is a 5 dimensional array of arrays (henceforth referred to as a tensor) in the shape [n, 24, 76, 151, 2], where n is the batch size, and is variable. 24 x 76 x 151 is the image size of the atomic structure of each amino acid residue in the batch represented as a 3D spherical grid. Finally the remaining vector size 2 contains the mass and charge for the respective amino acid (implying the 3D image already has a depth of 2 features).

This tensor is presented to the first convolutional layer, which applies 16 different 5 x 5 x 5 kernels to the 3D array representing the spherical grid and produces as many feature maps, also of 24 x 76 x 151 (size preserved through padding). Each feature map also has a bias offset added.

The ReLU activation function is then applied element wise over each of these 16 feature maps, initializing <0 values to 0.

The feature maps then undergo max-pooling, the maximums of 2 x 2 x 2 segments of the feature-maps are computed and kept, the other values are discarded, each of the feature maps are downscaled twofold in each dimension.

At this point the image is size 12 x 38 x 75.5 with a depth of 16 (feature maps).

This is then presented to the second convolution layer, which applies the 16; 5 x 5 x 5 kernels once more on this downscaled image. Again a bias offset is added to each feature map and ReLU is applied to each element of each feature map.

A second round of 2 x 2 x 2 max-pooling is undertaken.

At this point the image is depth 16 at size 6 x 19 x 37.75.

The image is then flattened into a 1D vector of size 4304 and input into a fully connected layer, with one hidden layer.

The hidden layer has 32 nodes, the number of connections between the input layer of the FCL and the hidden layer is 4304 x 32 = 137728.

the hidden layer then moves to an output layer of 21 nodes, the number of connections between these two layers in this transition is 672.

The output layer gives 21 possible classifications, which are the 20 amino acids, and 1 dummy class for failed classifications. Softmax is used to created a probability distribution with 21 points, showing the likelihood of the input data corresponding to each amino acid.

The format of the output data is an array shape [n , 21], where n is the batch size.

During training the number of weight parameters to optimise during training becomes 16 + 16 + 137728+ 672 = 138432 in total. The number of bias parameters is 16 + 16 +32 + 21 = 85.

training of these parameters is completed by minimizing the mean of cross entropy function over mini-batches using the Adam optimizer.

Adam is a more advanced optimization algorithm than gradient descent that allows for each parameter to have its own learning rate adapted separately, rather than a global learning rate for all parameters. Adam is specifically designed to be computationally efficient in handling sparse gradients on noisy problems. (Kingma and Ba, 2014)

As an advanced add-on, this model includes a ‘drop-out’ mechanism that allows some nodes to be randomly dropped along with their connections. This prevents units co-adapting too much, and prevents over-fitting to the training data. Over-fitting is explained more thoroughly in the discussion section. Figure 15 belowshows an illustration of how at each iteration of the training step, drop out can preserve the weights of some pathways. (Srivastava *et al.*, 2014)

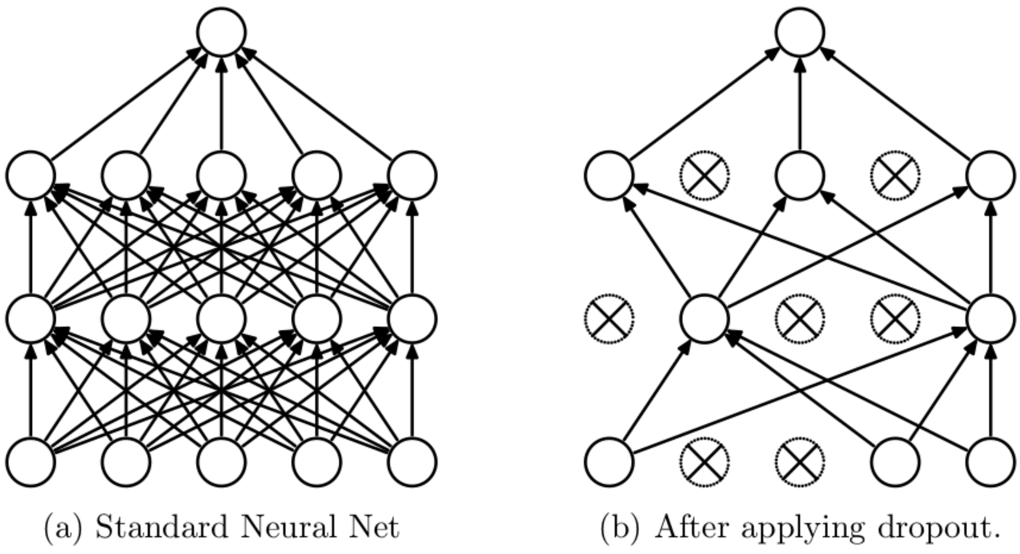


Figure 15. Above. Diagram illustrating the loss of weighted pathways observed when applying random drop-out to a training step Image adapted from (Srivastava et al., 2014)

Substitution to a better model

Due to time constraints and the limitations of the model developed by myself, toward the end of the study the model was substituted from the aforementioned one designed by myself, to a more complex model with more finely tuned hyper-parameters developed by Wouter Krogh Boomsma and Jes Frellsenthe.

During this study it was always intended that the models should be substituted, in the 8 week period it was unlikely I would create a superior model to the one that already existed, due to tailoring model architecture to suit a dataset and tweaking hyper-parameters being an arduous process as well as containing an element of trial and error. The process of making my own model was intended to be educational.

The model I created unfortunately didn't get higher than 10% training accuracy, and was never well enough developed in its infancy to reach a point where it was worth being validated on a test set. The model developed by Wouter Krogh Boomsma and Jes Frellsenthe has a validation accuracy consistently around 44-47% and has a number of other benefits in the form of advancements such as recognising that there is wrapping in the Azimuthal angle on the spherical grid for each amino acid and regularization mechanisms.

Results

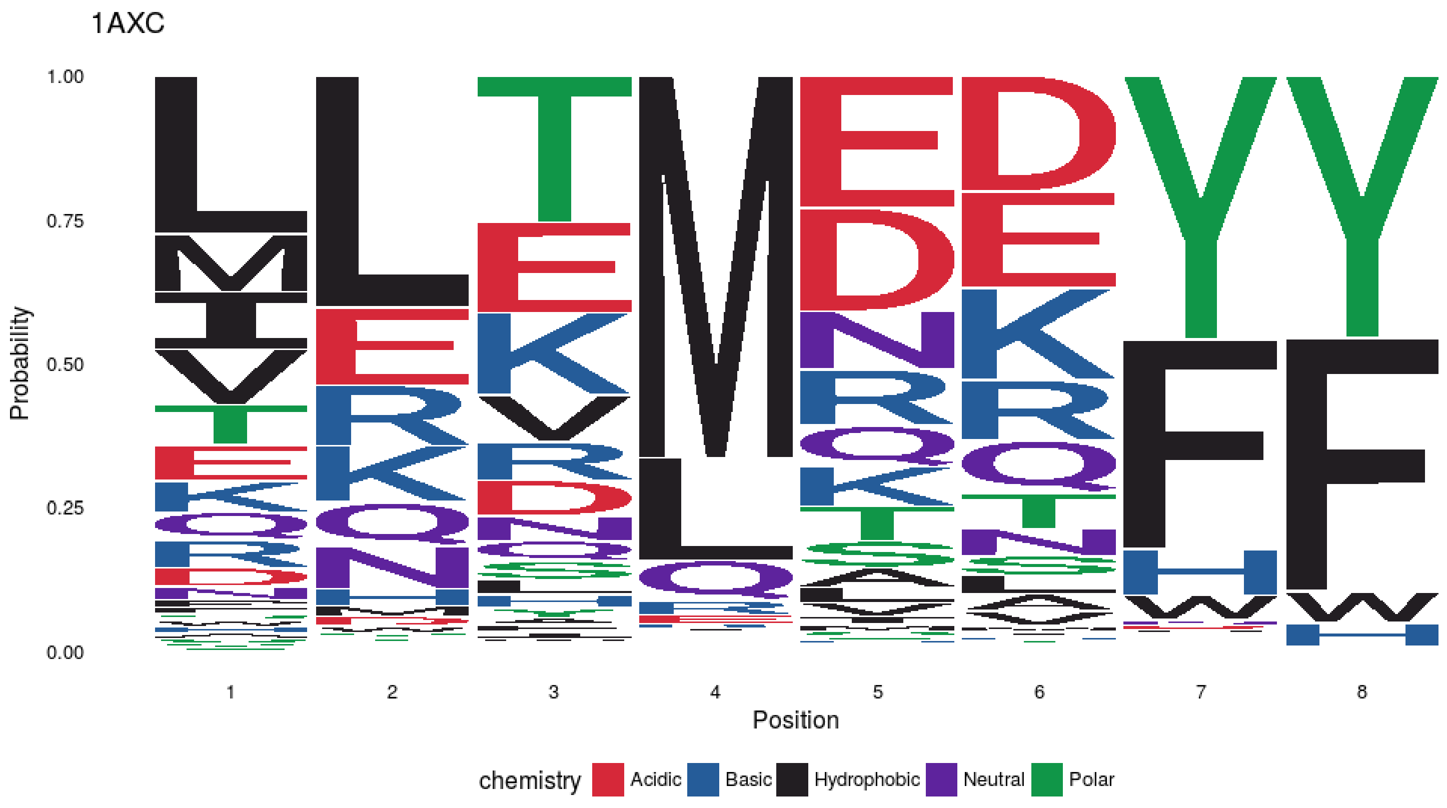


Figure 16. Above. Sequence logo graph generated from the probability distributions output for each amino acid in the 1AXC PIP-box SLiM by the convolutional neural-network.

For reference; the original sequence is QTSMTDFY in both structures and the PIP-box motif is Qxx[L/V/M]xx[F/Y][F/Y].

Figure 16 shows that the 1AXC prediction seems to have expressed most of motif seen in the PIP-box SLiM. Mostly positions 4, 7 and 8 fulfil this with confidence levels of methionine = 66%, tyrosine = 46% and tyrosine = 46% predicted at those respective sites. For positions 7 and 8 phenylalanine had similar confidence levels; 36% and 44% respectively. Leucine had a 18% confidence in being chosen for position 4, with is much less than the methionine, but much greater than all other amino acids for that position.

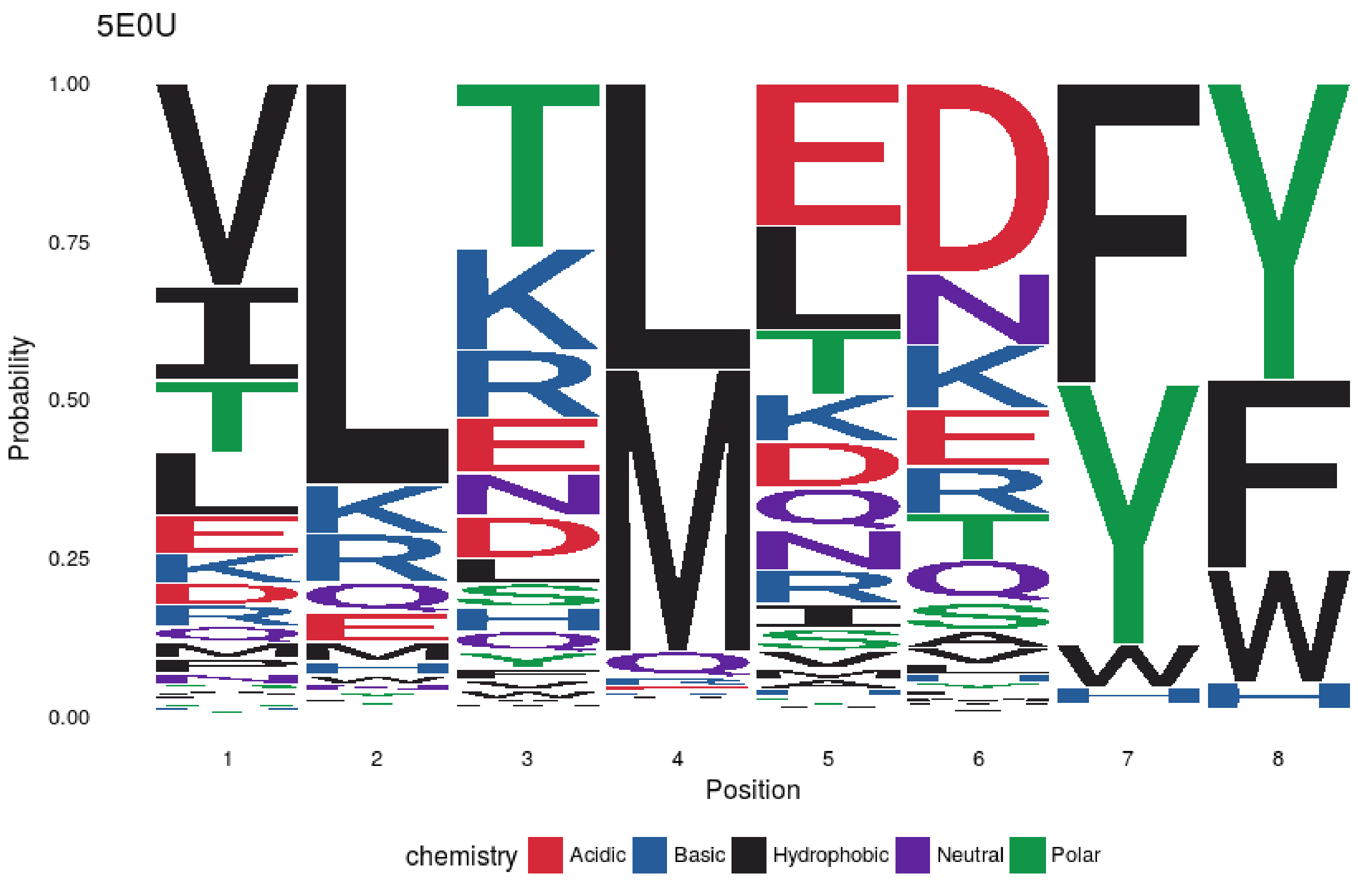


Figure 17. Above. Sequence logo graph generated from the probability distributions output for each amino acid in the 5E0U PIP-box SLiM by the convolutional neural-network.

The 5E0U prediction seem to have expressed most of the PIP-box SLiM motif also, again positions 4, 7 and 8 fulfil this. For position 4 the confidence levels are leucine = 45% and methionine = 44%. for position 7 ; 4 phenylalanine =48% and tyrosine = 41%. finally for position 8; tyrosine = 46%, and phenylalanine = 30 %.

In both examples the glutamate at position 1 is missed, the model seemingly favouring Hydrophobic residues at this position instead, notably leucine and valine, in 1AXC and 5E0U respectively. Although these predictions only held 0.27% and 0.32% confidence respectively.

Position 2 shows a preference for leucine in both examples predicted with 40% confidence for 1AXC and 64% 5E0U certainty by the model.

Position 3, 5, & 6, have no clear favourite in either structure, with many of the amino acids having similar levels of probability of them inhabiting that position. It should be noted though that D was slightly higher than its competitors for position 6 in both examples (0.20 confidence for A1XC and 0.30 for 5E0U), and that is congruent with the original sequence.

Acidic compounds, as a group seem to have been favoured slightly for positions 5 & 6, especially in 1AXC. Though collectively they only about 40% of the probability distribution available for those positions in 1AXC and 30% in 5E0U.

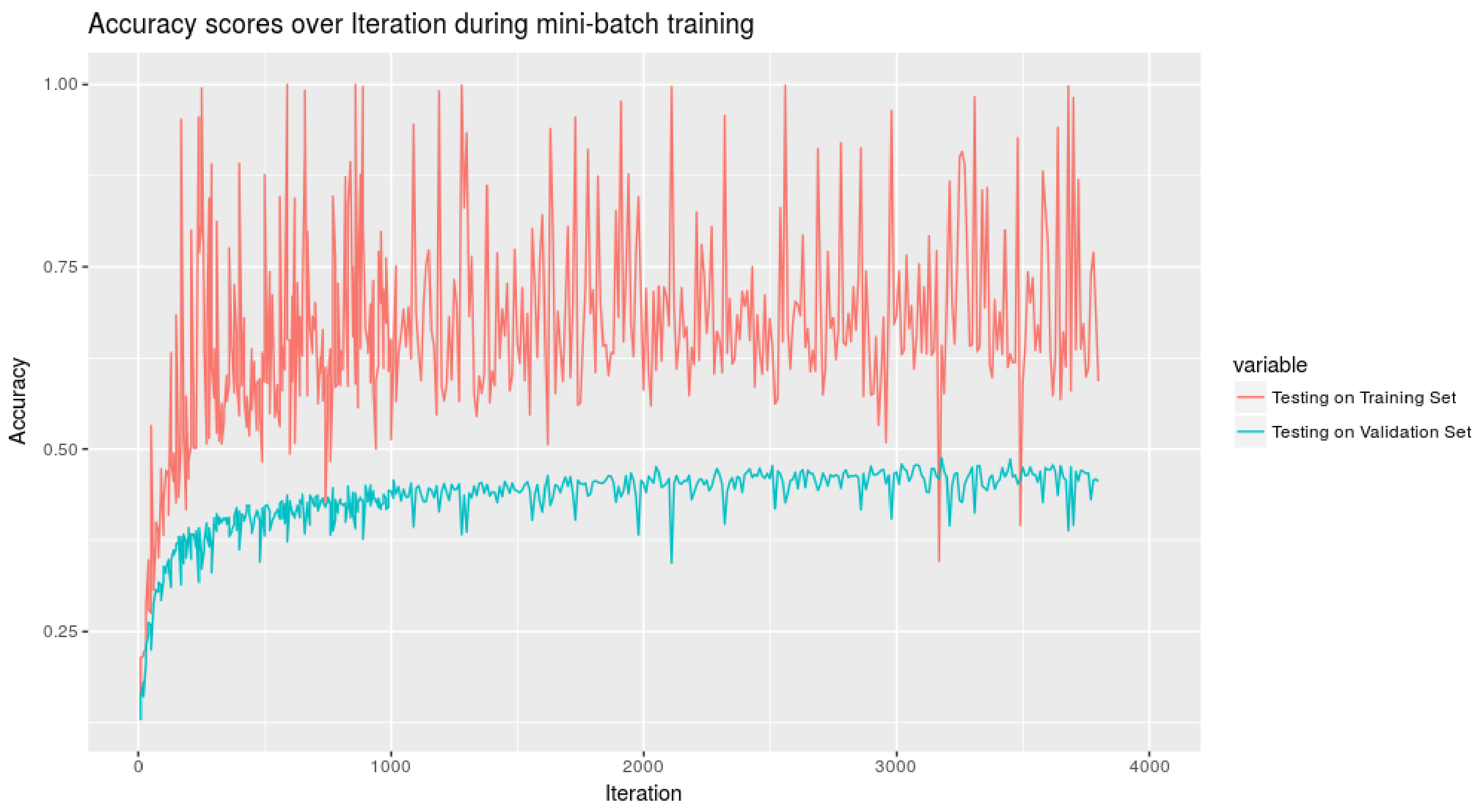


Figure 17. Above. Comparison of accuracy levels between the training set and the validation set over 3800 iterations of mini-batch training; parameters were adjusted and accuracy levels were assessed every 10th iteration.

Figure 17 shows that there is consistently a great deal of variance in the accuracy of the training set.

The training set can also be seen to have a consistently higher accuracy then the validation set.

The validation set seems to reach a ~45% in under 500 iterations, then slowly increases to ~47/48% over the next 3000 iterations.

There is no evidence to suggest continuing onward with more iterations would be beneficial or detrimental to the accuracy of the model on the validation set.

Discussion

The prediction results for both 1AXC and 5E0U seemed reasonable but need improvement. With the exception of the glutamate in position one being missed both times, the predicted structures matched the PIP-box motif. With regards to predicting the exact correct sequence, in both instances only 3/8 positions were correct, however there are only 4 positions which have any conserved sequence, and the model generally chose sensible classifications for these positions (again, except glutamate).

It is expect that the variable positions (2, 3, 5 & 6), would have much lower accuracy, as their conformation is much less likely to have appeared in the training set, thus making it harder for the model to recognise it.

With regard to glutamate; SLiMs are short and disordered by nature with only a few defined residues; as such the noise from stochastically recurring residues, has in past studies been found to be extensive, and has been found to be a prominent barrier to *De Novo* discovery of genuine motifs. It could be the case that something similar to this has affected the glutamate signal. (Bhowmick, Guharoy and Tompa, 2015; Edwards and Palopoli, 2015; Song and Gu, 2015)

It could be the case that our training data coincidentally has a lot of PIP-box motifs, without the canonical position 1 glutamate, due to them being of an irregular conformation. The nature of positions 4, 7 & 8 to be conserved, but still have one or two alternate possible residues may make them more easily detectable by the model.

Furthermore Position 2 in both structures showing preference for leucine is curious as there is no known reason why this may be the case, unless its a peculiarity represented in the training data.

It is far more likely however, that the deviations from the expected values observed in the results of this study are due to the low accuracy rating and difficulties of the model, and the resulting low confidence predictions. As can be seen from figure 14. the accuracy on the validation set never increases beyond 50%

Similarly only one prediction ever gave a confidence level of >50% (methionine 1AXC – 60%) This could also be due to the nature of the PIP-box having multiple acceptable possibilities for even its conserved regions.

The tests conducted here have neither confirmed nor denied the ability of this model to predict alternative conformations for SLiMs, where there are known alternatives for conserved structures, it has identified them, but no unorthodox predictions have been raised. This faculty would perhaps be better tested on larger data sets or other SLiMs of both more and fewer strictness in the conserved regions.

Figure 16 also shows that the accuracy of the training set is consistently higher than the validation set, seemingly by ~20% this is indicative of over-fitting, and suggests that the model has an inappropriately high level of complexity, it captures the variance in the training set so well, it cannot be as successfully applied more generally.

This can be counteracted using a few different methods, this model already has regularization step, but a drop-out could be implemented also, or the number of nodes utilized in the FCL could be lowered. In practice it can be delicate balancing act to find the perfect amount of ‘fit’.

As the accuracy seems to stop improving (or at least slow to an unmanageably small increase rate) after 500-1000 or so iterations, it could be useful to try using different optimization functions, or ensuring that the utilized optimization is not stuck in a local minimum. Finally investigating the effect the learning rate is having and considering altering the way this works could be beneficial, if the step size becomes too large as the model progresses, decaying the learning rate can help.

Improving the size and quality of the training data set will also always be beneficial to machine learning based techniques.

Testing the model further on sequences on both disordered sequences and sequences with more consistent structures, would perhaps allow for a more intuitive understanding of how successfully it can classify amino acids based on training. And testing it on a great many more sequences will allow more confident inference about its behaviour.

Future potential areas of investigation could include extending the model to be able to include the torsion bonds in the peptides would be challenging but would help identifying sequences where one or more amino acids in a sequence may infer their neighbours.

Extending the model beyond 3D image analysis completely to somehow also take into account the chemical properties of each residue, e.g. acidity, hydrophobic. So as to give another set of criteria to also classify by could be useful. Perhaps also an experiment in building neural networks that work in tandem, that would have further use in computational biology as well as other fields.

Conclusions

The model utilized here, after being trained managed to correctly identify 3 of 4 conserved positions, in an SLiM of 8 residues, paradoxically it consistently misclassified the only conserved, invariable region.

It is unclear whether the errors experienced have occurred because of poor training data, the nature of testing on inherently disordered sequences such as SLiMs, or are because the model requires further improvement.

It is unclear if or if not the model could be used to find new alternative conformations for SLiMs that would fulfil a similar role.

More rigorous testing should be undertaken so as to ascertain the the true efficacy of this model, and it should be widely applied to different sequences of varying levels of conservation.

From the limited analysis here; it seems further steps are needed to tweak the architecture and/or hyper-parameters to raise its efficacy to a level where it can be trusted to consistently give useful predictions. Regularization of a perceived over-fitting problem could be a good place to start.

Other avenues of expansion could include extending the model beyond single amino acid residues to examine whole peptides for torsion angles or other neighbouring phenomena to make inferences about local structure.

Acknowledgements

Wouter Krogh Boomsma and Jes Frellsenthe for the final model architecture used in this study and the optimisation of it’s hyper-parameters. As well as the python modules deepfold_batch_factory and deepfold_grid, which were integral to this study.

Special thanks to Wouter Krogh Boomsma especially for providing the necessary datasets and miscellaneous scripts to implement this project, as well as tutelage, time and patience in supervising the proceedings.

Citations

Berman, H. M. *et al.* (2000) ‘The Protein Data Bank’, *Nucleic Acids Research*. Oxford University Press, 28(1), pp. 235–242. doi: 10.1093/nar/28.1.235.

Bhowmick, P., Guharoy, M. and Tompa, P. (2015) ‘Bioinformatics Approaches for Predicting Disordered Protein Motifs’, in *Advances in experimental medicine and biology*, pp. 291–318. doi: 10.1007/978-3-319-20164-1_9.

Boomsma, W. and Frellsen, J. (2017a) ‘Deepfold_Batch_factory’, *Unpublished*.

Boomsma, W. and Frellsen, J. (2017b) ‘Deepfold_Grid’, *Unpublished*.

Colah (2015) *Calculus on Computational Graphs: Backpropagation -- colah’s blog*. Available at: https://colah.github.io/posts/2015-08-Backprop/ (Accessed: 27 August 2017).

Davey, N. E., Cyert, M. S. and Moses, A. M. (2015) ‘Short linear motifs – ex nihilo evolution of protein regulation’, *Cell Communication and Signaling*. Cell Communication and Signaling, 13(1), p. 43. doi: 10.1186/s12964-015-0120-z.

Dmcq (2012) *Spherical coordinate system, with notation for angles as used in physics*. Available at: https://commons.wikimedia.org/wiki/File:3D_Spherical.svg (Accessed: 27 August 2017).

Duffy, C. M., Hilbert, B. J. and Kelch, B. A. (2016) ‘A Disease-Causing Variant in PCNA Disrupts a Promiscuous Protein Binding Site’, *Journal of Molecular Biology*, 428(6), pp. 1023–1040. doi: 10.1016/j.jmb.2015.11.029.

Edwards, R. J. and Palopoli, N. (2015) ‘Computational Prediction of Short Linear Motifs from Protein Sequences’, in *Methods in molecular biology (Clifton, N.J.)*, pp. 89–141. doi: 10.1007/978-1-4939-2285-7_6.

Gulbis, J. M. *et al.* (1996) ‘Structure of the C-terminal region of p21(WAF1/CIP1) complexed with human PCNA.’, *Cell*, 87(2), pp. 297–306. Available at: http://www.ncbi.nlm.nih.gov/pubmed/8861913 (Accessed: 25 August 2017).

Kingma, D. P. and Ba, J. (2014) ‘Adam: A Method for Stochastic Optimization’. Available at: http://arxiv.org/abs/1412.6980 (Accessed: 28 August 2017).

Marabini, R. *et al.* (2017) ‘The PIP-box revisited. Protein intrinsic disorder and high pI prevail in interactions with proliferating csll nuclear antigen (PCNA).’, *Unpublished - Manuscript in preparation*, p. 29.

Moldovan, G.-L., Pfander, B. and Jentsch, S. (2007) ‘PCNA, the Maestro of the Replication Fork’, *Cell*, 129(4), pp. 665–679. doi: 10.1016/j.cell.2007.05.003.

Nguyen, D. *et al.* (2017) ‘Gender Recognition from Human-Body Images Using Visible-Light and Thermal Camera Videos Based on a Convolutional Neural Network for Image Feature Extraction’, *Sensors*. Multidisciplinary Digital Publishing Institute, 17(3), p. 637. doi: 10.3390/s17030637.

Pavlovsky, V. (2017a) *A Brief Introduction To Artificial Neural Networks*. Available at: https://www.vaetas.cz/blog/introduction-artificial-neural-networks/#relu-function (Accessed: 26 August 2017).

Pavlovsky, V. (2017b) *Introduction To Convolutional Neural Networks*. Available at: https://www.vaetas.cz/blog/intro-convolutional-neural-networks/ (Accessed: 26 August 2017).

Ponce, P. *et al.* (2014) ‘Brain Computer Interfaces for Cerebral Palsy’, in *Cerebral Palsy - Challenges for the Future*. InTech. doi: 10.5772/57084.

Song, T. and Gu, H. (2015) ‘Discovering short linear protein motif based on selective training of profile hidden Markov models’, *Journal of Theoretical Biology*, 377, pp. 75–84. doi: 10.1016/j.jtbi.2015.03.010.

Srivastava, N. *et al.* (2014) ‘Dropout: A Simple Way to Prevent Neural Networks from Overfitting’, *Journal of Machine Learning Research*, 15, pp. 1929–1958. doi: 10.1214/12-AOS1000.

Stoimenov, I. and Helleday, T. (2009) ‘PCNA on the crossroad of cancer’, *Biochemical Society Transactions*, 37(3), pp. 605–613. doi: 10.1042/BST0370605.

Ujjwalkarn (2016) *An Intuitive Explanation of Convolutional Neural Networks*. Available at: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.

Wagih, O. (2017) ‘ggseqlogo: a versatile R package for drawing sequence logos’. Available at: https://doi.org/10.1093/bioinformatics/btx469.

Wells, J. A. and McClendon, C. L. (2007) ‘Reaching for high-hanging fruit in drug discovery at protein?protein interfaces’, *Nature*, 450(7172), pp. 1001–1009. doi: 10.1038/nature06526.