A Deep Learning Model for Predicting Tumor Suppressor Genes and Oncogenes from PDB Structure : An implementation

Hiremath,Yashas
University of Pittsburgh
yah62@pitt.edu

Manzoor,Shanim
University of Pittsburgh
shm150@pitt.edu

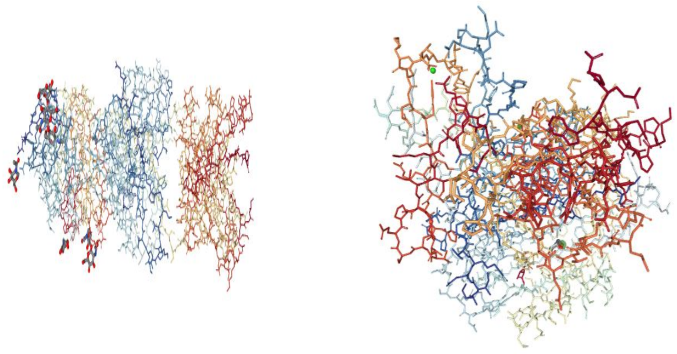
Abstract

The aim of the project is to re-implement a novel research paper that aimed to use a convolutional neural network(CNN) to classify between two of the most common gene mutations, namely the oncogenes and tumor suppressor genes after generating data using the Protein Data Bank(PDB). Using the amino acid properties (20 of them) , feature maps were generated along with the co-ordinates on the alpha carbon atoms.The dataset was divided into 3 parts for co-ordinate spaces and sparsified to showcase real data . Once this was done, the parallel neural network showed a best case Area under ROC of 0.65. The re-implementation of the author's code using Python and Pytorch proved to be succesful but with lesser accuracy on the updated dataset.

1 Introduction

Cancer comes across the biological space in the molecular level in many different types including mutations to genes which would usually suppress cancer growth causing them to be inhibited or in other cases mutations genes which are activated and cause much higher levels of cancerous cell growth. the accurate prediction of a disease outcome is one of the most interesting and challenging tasks for physicians. As a result, ML methods have become a popular tool for medical researchers[1]. Multitude of quality research has been done for various purposes such as prediction of cancer using different indicators as well as finding binding sites[2].

The rapid advancement of machine learning and especially deep learning continues to fuel the medical imaging community's interest in applying these techniques to improve the accuracy of cancer screening[3], As such the route taken by this research piece was to automatically detect one of two major gene mutations connected with cancerous growth. Using the biochemical features coupled with the structure of the related proteins could prove to be very useful in targeting these agents while



developing new treatments.

Figure 1:The figure on the left is of the 3B2U protein structure which has been pre-classified as an oncogene while the one on the right is of the 4L29 which is of the Tumor suppressor gene type.

Tumor suppressor genes (TSGs) play a major role in the carcinogenic process by manipulating growth of cells and apoptosis, inhibiting the formation of tumors. Mutations in TSGs inactivate their inhibitory function, thereby contributing to the carcinogenic process. Proto-oncogenes likewise are involved in cell growth; when mutated, these oncogenes (OGs) promote cancer through proliferation of cells. Unlike TSGs which require a double hit to inactivate the gene, mutations to OGs are dominant with one copy of the gene needing to be mutated to promote cancer[4].

OG and TSG can also be fused together between themselves or in other cases fused with non-malicious genes In research previously done by Osborne[5] cancer identification performance was seen to be improved.The research done by Tavanei et al [6] used a deep convolutional neural network(CNN) to classify TSGs and OGs. The reason they selected it was due to CNNs proven high performance in visual feature extraction and classification in multiple research papers as well as their ability to find useful information from PDB structures by mapping biochemical features while looking at alpha atoms.

2 Background-Dataset Preparation

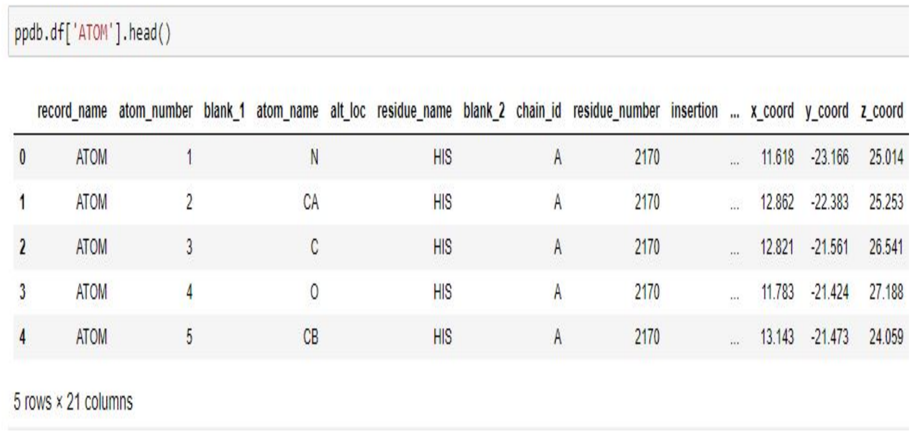
Essentially to prepare the dataset , the procedure while convoluted requires data from various sites and preprocessing to get ready before feeding into the neural network. Firstly, a comma separated values (CSV) file is exported using the COSMIC(Catalogue Of Somatic Mutations In Cancer) V90 data instead of the V82 that was used in the previous implementation. After weeding out the 'Role in Cancer ' Column to filter out only those genes which are purely OGs or TSGs. Finally the annotated gene lists cumulatively indicate 182 Tumor Suppressor genes and 106 Oncogenes. Using this list of Gene symbols which are unique to these genes , mapping of PDB entries is done to the gene symbols which are also unique. Essentially these PDB entries are analogous to Ensembl ids.

The PDB entries are then mapped to the multiple PDB Id's that they may be connected to using the Uniprot web tool(http://www.uniprot.org). This list can then be used to download the PDB structures from the protein data bank(www.rcsb.org/pdb/download/download). The structures selected are those of X-ray or NMR modalities.

```
1: $C_\alpha$.inds(x,y,z) = atom.select(pdb, "$C_\alpha$")
2: Gravity.inds(x,y,z) = mean($C_\alpha$.inds(x,y,z))
3: Polar.inds = polar($C_\alpha$.inds(x,y,z) - Gravity.inds(x,y,z))
4: Surface.atoms = selectSur(Polar.inds)
5: Surface.cartesian = mapCatesian(Surface.atoms)
6: Surface.cartesian.norm =
       Round(2$\times$(Surface.cartesian-min(Surface.cartesian)))
7: Property(x,y,z) = property(Surface.cartesian.norm)
```

Figure 2: Psuedo-Code for PDB feature Extraction[6]

These PDB files have different type of records such as ATOM,HETATM,HELIX etc. of which ATOM is selected.The alpha carbon atoms on the surface are selected with the highest radius based on the psuedocode detailed above. A single row(not technically a row) but using *BioPandas*(https://rasbt.github.io/biopandas) , a PDB structure can be automatically converted to a user readable dataframe. From the carbon atoms selected(alpha) using Element and chain identifier columns as well as the radius after finding the max distance from a reference point using the co-ordinates(x,y and z) further filtering is done.



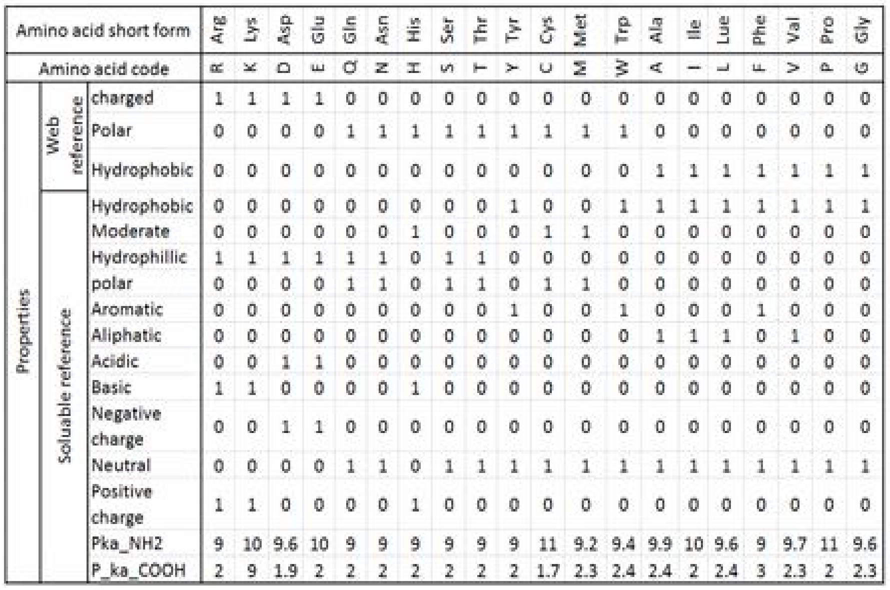
```
ppdb.df['ATOM'].head()
```

```
   record_name  atom_number  blank_1  atom_name  alt_loc  residue_name  blank_2  chain_id  residue_number  insertion  ...  x_coord  y_coord  z_coord
0         ATOM            1                   N                    HIS                  A            2170             ...   11.618  -23.166   25.014
1         ATOM            2                  CA                    HIS                  A            2170             ...   12.862  -22.383   25.253
2         ATOM            3                   C                    HIS                  A            2170             ...   12.821  -21.561   26.541
3         ATOM            4                   O                    HIS                  A            2170             ...   11.783  -21.424   27.188
4         ATOM            5                  CB                    HIS                  A            2170             ...   13.143  -21.473   24.059

5 rows × 21 columns
```

Figure 3: The figure shows a snapshot of a PDB structure's first 5 rows after loading using biopandas

Once this is done, each PDB (after filtering) is converted to a CSV file with the following rows : Residuename(amino acid) ,x ,y and z co-ordinates. The amino acid can correspond to 22 different types of which 20 of them have biochemical properties already researched by Tavanei et al[6] . There are 2 others including UNK (Unknown) which hasn't been defined by them and won't be considered for this training set and have been filtered out further . After all of this , mapping is done according to the amino acid based on their biochemical proprties and tacked on to the CSV file where the dataset ends up looking like the figure below.

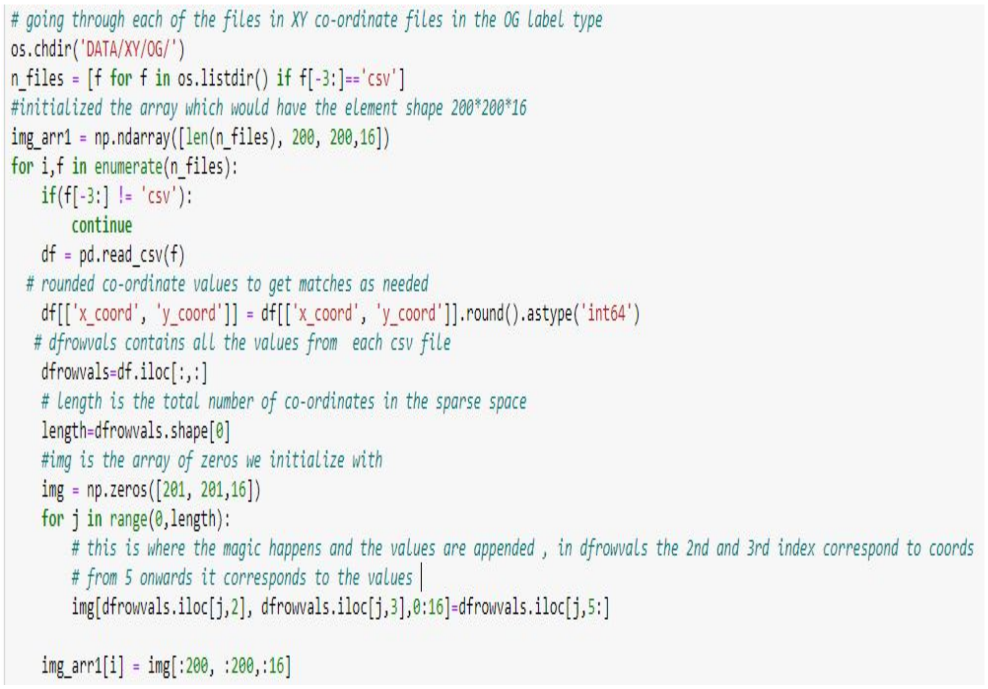


Amino acid short form			Arg	Lys	Asp	Glu	Gln	Asn	His	Ser	Thr	Tyr	Cys	Met	Trp	Ala	Ile	Lue	Phe	Val	Pro	Gly
Amino acid code			R	K	D	E	Q	N	H	S	T	Y	C	M	W	A	I	L	F	V	P	G
Properties	Web reference	charged	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Polar	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
		Hydrophobic	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
	Soluable reference	Hydrophobic	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1	1	1
		Moderate	0	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0
		Hydrophillic	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0
		polar	0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	0	0	0	0
		Aromatic	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0
		Aliphatic	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0
		Acidic	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Basic	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
		Negative charge	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Neutral	0	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
		Positive charge	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
		Pka_NH2	9	10	9.6	10	9	9	9	9	9	9	11	9.2	9.4	9.9	10	9.6	9	9.7	11	9.6
		P_ka_COOH	2	9	1.9	2	2	2	2	2	2	2	1.7	2.3	2.4	2.4	2	2.4	3	2.3	2	2.3

Figure 4: Amino acids vs Biochemical properties

The next step is to understand how to feed this unstructured data into a neural network in terms of shape and context. With this in tow, the idea is to diferentiate into different groups depending on labels ,namely 'TSG' and 'OG' but also considering that the data is sparse since looking at the number of co-ordinates that have values there are only between 100-250 in each case. To create a sparse dataset , the first step was to scale the co-ordinates to the range from 0-200 for ease and to conform with Tavanei et al [6] in terms of their execution. Next , an essential part was to separate the co-ordinate space from X-Y-Z space to 3 spaces X-Y,Y-Z and X-Z . For this we created 3 separate directories according to these spaces and had them under two categories (the labels) . Once done each datapoint or PDB ID would have the co-ordinates and the 16 biochemical features associated with it. For the population of dummy co-ordinates over a 200x200 space ,we (1) initialize an image array with each element having the shape 200x200x16 and for each csv file . Next (2) while traversing each co-ordinate of the array wherever the match happens between co-ordinates and the original dataframe's co-ordinates , we append the 16 features. (3) This final array is then saved before doing the same for the other 5 sets as well . Overall the data looks to be of the shape :

- OG :1440*200*200*16 for each of the 3 sets
- TSG:1475*200*200*16 for each of the 3 sets

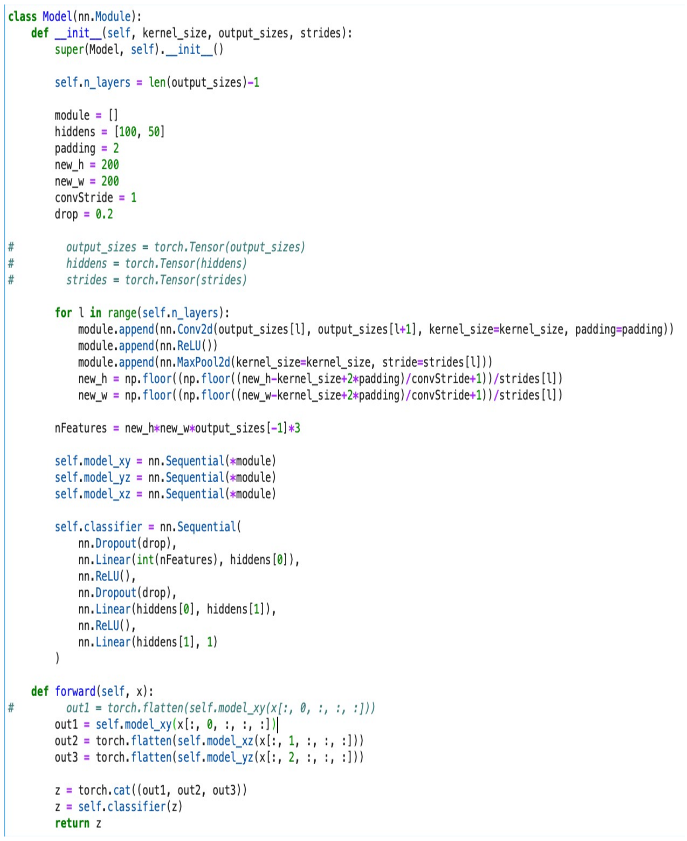


```python
# going through each of the files in XY co-ordinate files in the OG label type
os.chdir('DATA/XY/OG/')
n_files = [f for f in os.listdir() if f[-3:]=='csv']
#initialized the array which would have the element shape 200*200*16
img_arr1 = np.ndarray([len(n_files), 200, 200,16])
for i,f in enumerate(n_files):
    if(f[-3:] != 'csv'):
        continue
    df = pd.read_csv(f)
  # rounded co-ordinate values to get matches as needed
    df[['x_coord', 'y_coord']] = df[['x_coord', 'y_coord']].round().astype('int64')
   # dfrowvals contains all the values from  each csv file
    dfrowvals=df.iloc[:,:]
    # length is the total number of co-ordinates in the sparse space
    length=dfrowvals.shape[0]
    #img is the array of zeros we initialize with
    img = np.zeros([201, 201,16])
    for j in range(0,length):
        # this is where the magic happens and the values are appended , in dfrowvals the 2nd and 3rd index correspond to coords
        # from 5 onwards it corresponds to the values
        img[dfrowvals.iloc[j,2], dfrowvals.iloc[j,3],0:16]=dfrowvals.iloc[j,5:]

    img_arr1[i] = img[:200, :200,:16]
```

Figure 5: code snippet for co-ordinate matrix creation

3 Methods:Deep Learning Model- Architecture



```python
class Model(nn.Module):
    def __init__(self, kernel_size, output_sizes, strides):
        super(Model, self).__init__()

        self.n_layers = len(output_sizes)-1

        module = []
        hiddens = [100, 50]
        padding = 2
        new_h = 200
        new_w = 200
        convStride = 1
        drop = 0.2

#         output_sizes = torch.Tensor(output_sizes)
#         hiddens = torch.Tensor(hiddens)
#         strides = torch.Tensor(strides)

        for l in range(self.n_layers):
            module.append(nn.Conv2d(output_sizes[l], output_sizes[l+1], kernel_size=kernel_size, padding=padding))
            module.append(nn.ReLU())
            module.append(nn.MaxPool2d(kernel_size=kernel_size, stride=strides[l]))
            new_h = np.floor((np.floor((new_h-kernel_size+2*padding)/convStride+1))/strides[l])
            new_w = np.floor((np.floor((new_w-kernel_size+2*padding)/convStride+1))/strides[l])

        nFeatures = new_h*new_w*output_sizes[-1]*3

        self.model_xy = nn.Sequential(*module)
        self.model_yz = nn.Sequential(*module)
        self.model_xz = nn.Sequential(*module)

        self.classifier = nn.Sequential(
            nn.Dropout(drop),
            nn.Linear(int(nFeatures), hiddens[0]),
            nn.ReLU(),
            nn.Dropout(drop),
            nn.Linear(hiddens[0], hiddens[1]),
            nn.ReLU(),
            nn.Linear(hiddens[1], 1)
        )

    def forward(self, x):
#         out1 = torch.flatten(self.model_xy(x[:, 0, :, :, :]))
        out1 = self.model_xy(x[:, 0, :, :, :])
        out2 = torch.flatten(self.model_xz(x[:, 1, :, :, :]))
        out3 = torch.flatten(self.model_yz(x[:, 2, :, :, :]))

        z = torch.cat((out1, out2, out3))
        z = self.classifier(z)
        return z
```

Figure 6: Model Snapshot(code using PyTorch library)

The inputs provided are obtained from the final step of the previous efforts. Since these datasets are huge (8GB) each , we use the clusters provided by the Center of Research Computing(CRC) with their higher RAM and robust GPU support to enable the running of the model. Essentially there are 3 branches in the neural network for each of the projection spaces XY,YZ and XZ . Based on Tavanei's previous best performing model parameters , the kernel size, strides , hidden layer neuron count and the padding to be done are all setup. Since this is a binary classification model as well as there being a equitable distribution , the output classifier is straightforward. The activation function used is ReLU (Rectified Linear Unit) in the network. Dropout is used to increase robustness of the network .There are a total of 4 convolution and max pooling layers which act as visual feature extractors.

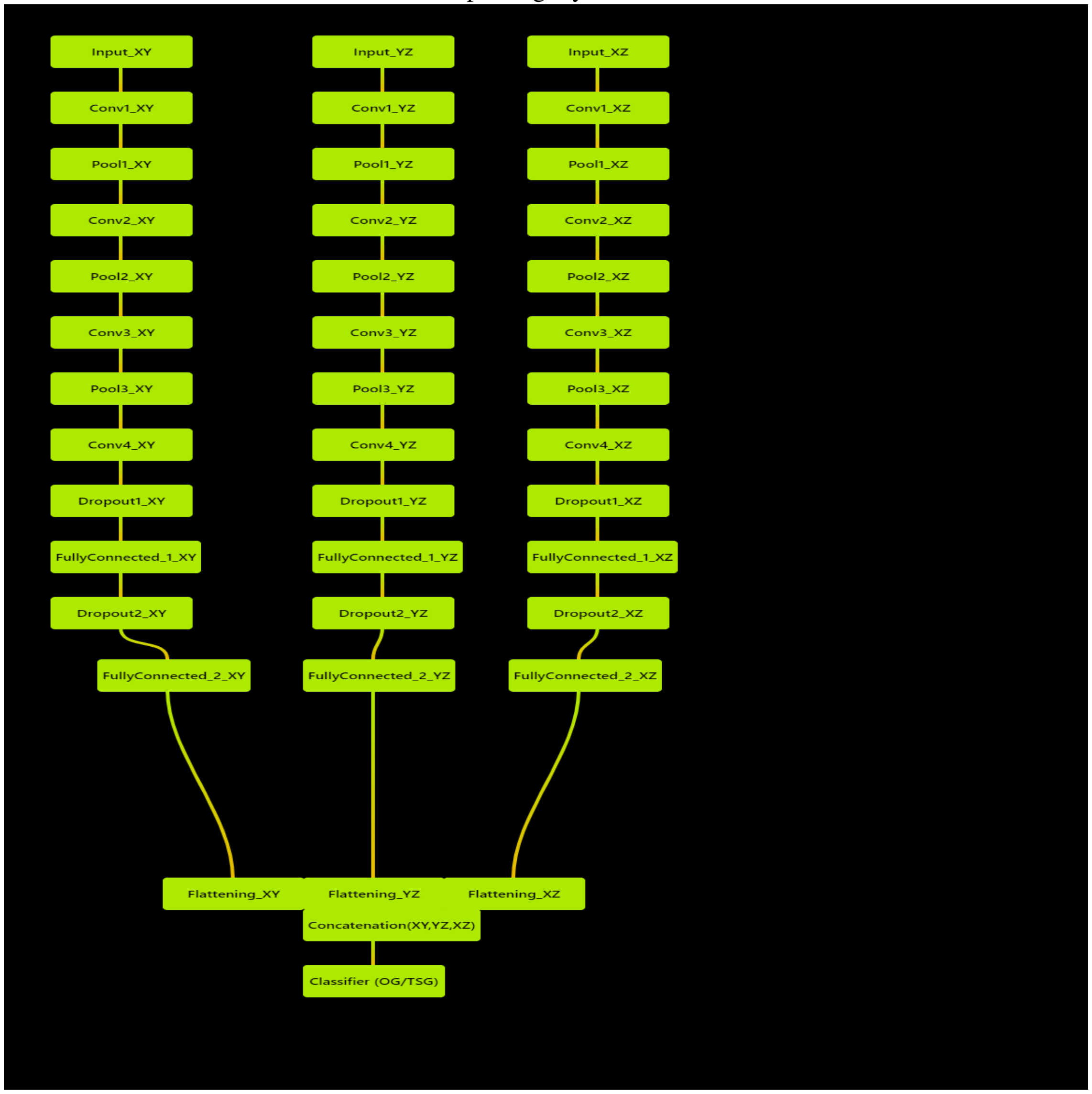
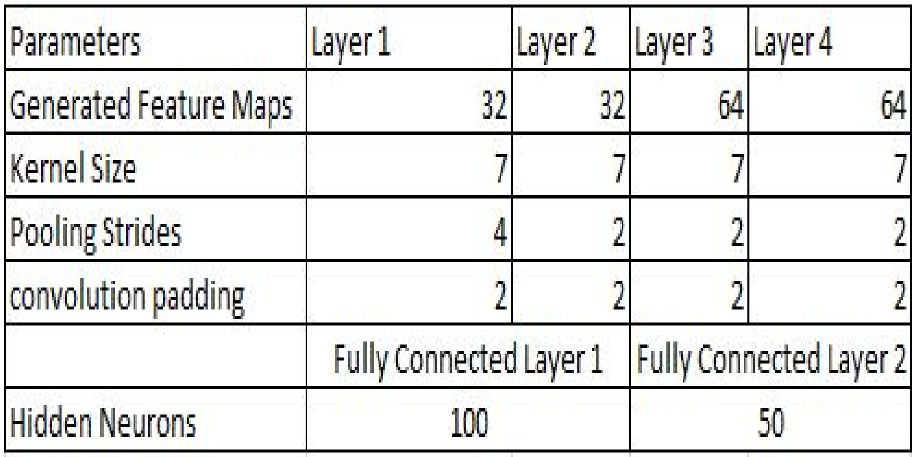


Figure 7: Architectural Overview: Parallel Convolutional Neural Network

Once the convolution and pooling step(total of 4 repetitions) are completed , the fully connected layers takes these visual features as inputs with the added benefit of dropout . At this point there are still 3 parallel networks running for the 3 co-ordinate spaces. From this point on flattening is done for each of the outputs before concatenating them into one array and feeding to the output classifier. Classifier consists of two fully connected layers followed by an output layer. The model parameters used are tabulated below :

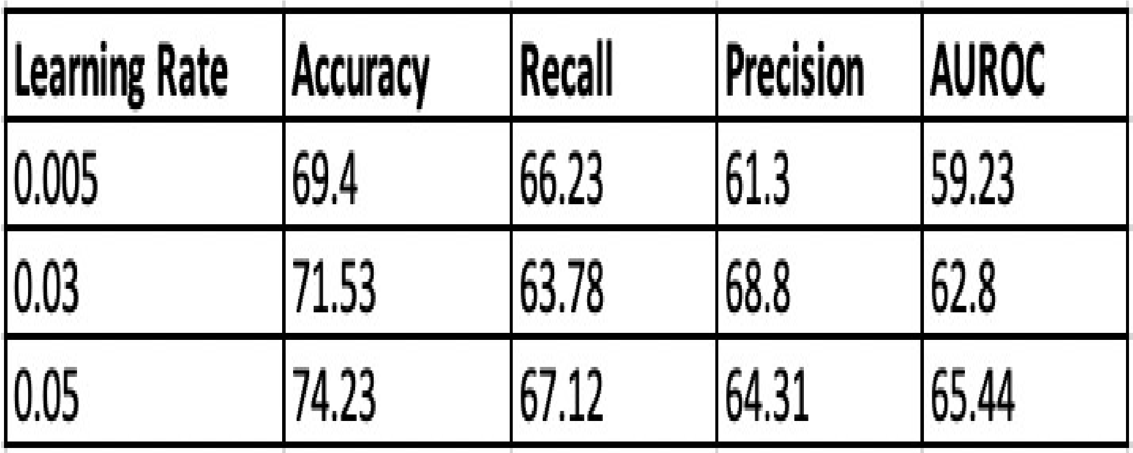


Parameters	Layer 1	Layer 2	Layer 3	Layer 4
Generated Feature Maps	32	32	64	64
Kernel Size	7	7	7	7
Pooling Strides	4	2	2	2
convolution padding	2	2	2	2
	Fully Connected Layer 1		Fully Connected Layer 2	
Hidden Neurons	100		50	

Figure 8: Model Parameters:Parallel Convolutional Neural Network

4 Results

After training the model on 3 different learning rates, 0.05,0.005 and 0.03 , the best results were seen with a learning rate of 0.05. The accuracy was lower than what the original research postulated. The tabulation of results observed similar to the original findings are shown below:



Learning Rate	Accuracy	Recall	Precision	AUROC
0.005	69.4	66.23	61.3	59.23
0.03	71.53	63.78	68.8	62.8
0.05	74.23	67.12	64.31	65.44

Figure 9: Model Evaluation: Results

With this there are a few caveats to note, namely *test set is smaller*! in our case. An issue we ran into and weren't able to clear towards the end was memory since our data is clearly sparse and loading it into memory for test evaluation lead to memory errors even on the cluster. On the smaller set of around 500 samples from the original 1000 that had been left for validation. As discussed during the original presentation, there were doubts on how the accuracy might not be realistic ,while we've gotten lower numbers, this may be due to a smaller dataset and less training attempts(learning rates).

Acknowledgments

- The support and practice from assignments detailed in the course by Prof. David Koes and Prof. Maria Chikina to be able to implement this project
- CRC for being able to use their resources to undertake the demands of the large dataset

References

[1]Konstantina Kourou, Themis P. Exarchos,Konstantinos P.Exarchos,Michalis V.Karamouzis,Dimitrios I.Fotiadis,Machine learning applications in cancer prognosis and prediction
Computational and structural biotechnology journal, ISSN: 2001-0370, Vol: 13, Page: 8-17

[2] C. Sotiriou, S.-Y. Neo, L. M. McShane, E. L. Korn, P. M. Long, A. Jazaeri, P. Martiat, S. B. Fox, A. L. Harris, and E. T. Liu, "Breast cancer classification and prognosis based on gene expression profiles from a population-based study," *Proceedings of the National Academy of Sciences, vol. 100, no. 18, pp. 10 393–10 398, 2003.Google Scholar*

[3] Shen, L., Margolies, L.R., Rothstein, J.H. et al. Deep Learning to Improve Breast Cancer Detection on Screening Mammography. *Sci Rep 9, 12495 (2019) doi:10.1038/s41598-019-48995-4*

[4] Slattery, M. L., Herrick, J. S., Mullany, L. E., Samowitz, W. S., Sevens, J. R., Sakoda, L., Wolff, R. K. (2017). The co-regulatory networks of tumor suppressor genes, oncogenes, and miRNAs in colorectal cancer. *Genes, chromosomes cancer, 56(11), 769–787. doi:10.1002/gcc.22481*

[5] C. Osborne, P. Wilson, and D. Tripathy, Oncogenes and tumor suppressor genes in breast cancer: potential diagnostic and therapeutic applications,*The oncologist, vol. 9, no. 4, pp. 361–377, 2004.*

[6] Tavanaei Amirhossein, Anandanadarajah Nishanth, Anthony Maida, and Rasiah Loganantharaj, "A Deep Learning Model for Predicting Tumor Suppressor Genes and Oncogenes from PDB Structure",*doi: 10.1101/177378, bioRxiv, 2017*