7 MUSCULAR DYSTROPHY DISEASE IDENTIFICATION THROUGH DEEP LEARNING APPROACH

This chapter demonstrates the self taught learning approach employed in building the muscular dystrophy disease identification model using a TensorflowDeepNeuralNetwork algorithm. In this work, deep learning approach is implemented using scikitflow, a machine learning framework coded in python on tensorflow environment. The beginning of the chapter gives an introduction about tensorflow, and then followed by the implementation of disease identification models on deep neural network through nucleotide and codon mapping schemes, comparison of shallow net and deep net in later sections.

Self-taught learning attempts automatically learn good features or representations based on training data. A deep neural network approach learn complex and abstract features automatically from unlabelled data and is employed in this research for identifying this genetic disease. Muscular dystrophy disease identification problem is modeled as a classification problem with self-extracted features, through deep neural network using Scikitflow in Tensor Flow environment. The benefit of positional cloning approach in this experiment supports generating the mutated gene sequences as portrayed in Chapter 4.

The trick to use deep learning models is to represent biological data, where in this work, 1000 diseased gene sequences are converted into 1-D representation in two schemes such as nucleotide mapping and codon mapping. The data set with instances related to five categories of muscular dystrophy that is Duchenne muscular dystrophy, Becker's muscular dystrophy, Emery-Dreifuss, Limb-girdle muscular dystrophy and Charcot Marie Tooth disease has been encoded and deepnet models have been implemented. Deep neural network, which is a kind of feedforward neural network, is used to build the muscular dystrophy disease identification model. The disease gene sequences were converted into a numpy array based on the hardcoded values and directly fed into tensorflow. The Tensorflow linear classifier and Tensorflow Deep neural network are employed and their parameters were tuned to attain good results.

7.1 TensorFlow

Tensorflow is an open source software library for numerical computation using data flow graphs. The mathematical operations are represented in the form of nodes in the graph, where the tensors are multidimensional data arrays which are the edges communicated between them. This architecture enables a distributed architecture where one or more CPUs are connected to a network in a single API. Tensor Board is a supporting tool given by TensorFlow to handle the in-depth visualization of the model. TensorFlow supports parallelization over multiple processors using CPUs or GPUs of the single machine.

Tensorflow environment setup is done in windows operating system by installing it through docker. The installation is made by connecting the windows machine and virtual machine using docker. Python is a programming language to run in the tensorflow environment. In this research work, the host machine is connected with the docker machine and the parallelization is achieved by building disease identification models using Scikit flow in jupyter notebook.

TensorFlow provides a good backbone for building different shapes of machine learning applications. Data mining and Predictive analytics is made simple by using the scikitflow interface in TensorFlow environment. Different shapes of ML models is built with single line machine learning code using scikitflow. TensorFlow APIs make comfortable to fit or predict models using deep learning approach. Scikitflow is a combination of scikitlearn and tensorflow and therefore the GridSearch and Pipeline are used to tune the parameters achieve machine learning in distributed framework.

Scikit-learn + TensorFlow = Scikit Flow

The machine learning in scikitflow is implemented in the Tensorflow 0.8.0 version that includes the major packages such as estimators, pandas, contrib.learn and dataframes. Training and testing on a deep neural classifier is done through TensorFlowDNNClassifier by explicitly specifying the number of nodes and the number of hidden layers.

7.2 Disease Identification Model using Deep Learning With Nucleotide Mapping Scheme

The first deep learning model aims in building the disease identification model by encoding the gene sequences using nucleotide mapping scheme. Features are extracted automatically and disease identification model is built using TensorflowDeepneuralnetwork classifier.

Nucleotide mapping

Gene sequence is a chain of categorical values. In the nucleotide mapping scheme, a genome sequence is considered as a fixed length 1D sequence window with four channels (A, G, C, T). Numerical representation of DNA sequences is necessary to apply a wide range of mathematical tools, including most of signal processing. In the decimal system representation T is encoded as 0, C is encoded as 1, A is encoded as 2 and G is encoded as 3. In this nucleotide mapping each base pair of the gene sequence is represented by its corresponding numeric code. Python script is written for the representation of nucleotides into integer values.

```
import numpy as np
def freq_numpy(dna_list):
frequency_matrix = np.zeros((4, len(dna_list[0])), dtype=np.int)
base2index = {'A': 0, 'C': 1, 'G': 2, 'T': 3}
for dna in dna_list:
for index, base in enumerate(dna):
frequency_matrix[base2index[base]][index] += 1
return frequency_matrix
```

Consider LMNA gene sequence, for example

```
>gi|383792147|ref|NM_170707.3| Homo sapiens lamin A/C (LMNA), transcript variant 1,
mRNAATGGAGACCCCGTCCCAGCGGCGCGCCACCCGCAGCGGGGCGCAGGCCAGCTC
CACTCCGCTGTCGCCCACCCGCATCACCCGGCTGCAGGAGAAGGAGGACCTGCAGGA
GCTCAATGATCGCTTGGCGGTCTACATCGACCGTGTGCGCTCGCTGGAAACGGAGAAC
GCAGGGCTGCGCCTTCGCATCACCGAGTCTGAAGAGGTGGTCAGCCGCGAGGTGTCC
GGCATCAAGGCCGCCTACGAGGCCGAGCTCGGGGATGCCCGCAAGACCCTTGACTCA
GTAGCCAAGGAGCGCGCCCGCCTGCAGCTGGAGCTGAGCAAAGTGCGTGAGTTTAAG
GAGCTGAAAGCGCGCAATACCAAGAAGGAGGGTGACCTGATAGCTGCTCAGGCTCGG
CTGAAGGACCTGGAGGCTCTGCTGAACTCCAAGGAGGCCGCACTGAGCACTGCTCTC
AGTGAGAAGCGCACGCTGGAGGGCGAGCTGCATGATCTGCGGGGCCAGGTGGCCAAG
CTTGAGGCAGCCCTAGGTGAGGCCAAGAAGCAACTTCAGGATGAGATGCTGCGGCGG
GTGGATGCTGAGAACAGGCTGCAGACCATGAAGGAGGAACTGGACTTCCAGAAGAAC
ATCTACAGTGAGGAGCTGCGTGAGACCAA
```

After applying the nucleotide mapping the corresponding gene sequence is translated into array of numeric values as shown below,

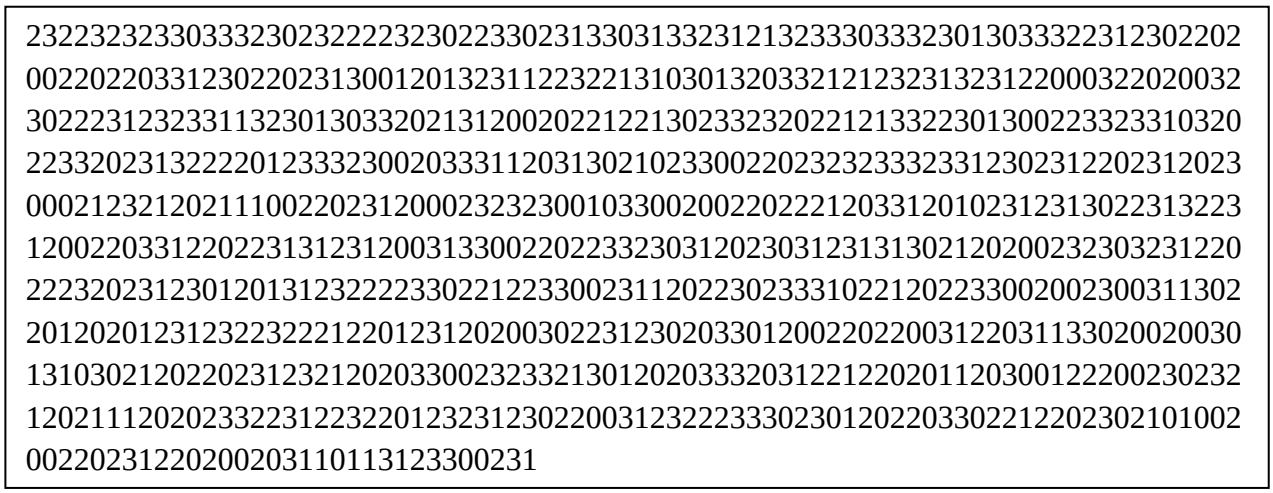

Fig.7.1 Translating Nucleotide into Numerical Values

Building the model

The 1-D array of nucleotides are converted into numpy arrays, as scikit – learn library accepts only a numpy array in its implementation. The data frame is built with the numpy array. For the unlabelled data used in the pre-training process, the numerically mapped sequences with decimal values are combined and the label information is removed to obtain the unlabelled data. Appropriate hyper-parameters such as the number of units per layer, learning rate and the epochs of iteration are determined for learning the deep neural network. The algorithm performs better if the sizes for all the hidden layers are closely same.

When the neural network becomes deeper, the feature level becomes higher. So a large amount of unlabelled data is expected to contribute to the feature detector. To explore the effect of the unlabelled data set size on our method's performance, the corresponding experiments are conducted. Considering both accuracy and efficiency n-layered DNN architecture with number of hidden units ranging from 20 to 100 are framed. The number of hidden layers 'n' is varied from 3 to 8, and the results are observed. The deep learning method used in this work was the fully connected multi-layer perceptron with 1000 input nodes for disease gene sequence with nucleotide and codon mapping schemes. The experiment shows that the best combination of parameters was 3 hidden layers with 70, 80 and 70 in each with sigmoid activation function.

With 0.5 as dropout rejection rate the DNN was tested for 20% and 80% of sequences at each layer. The script for building the classifier is shown below

Script for Tensorflow Linear Classifier:

```
from numpy import genfromtxt
my_data = genfromtxt('deep_new1_1.csv', delimiter=',')
from sklearn.cross_validation import train_test_split
X = my_data[:,-1]
y = my_data[:,1]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,rand
om_state=0)
classifier = skflow.TensorFlowLinearClassifier(n_classes=5)
classifier.fit(X_train,y_train)
score = metrics.accuracy_score(y, classifier.predict(X))
print("Accuracy: %f" % score)
```

Script for TensorflowDeepNeuralNetworkClassifier

```
from numpy import genfromtxt
my_data = genfromtxt('deep_new1.csv', delimiter=',')
from sklearn.cross_validation import train_test_split
X = my_data[:,-1]
y = my_data[:,1]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
from sklearn import metrics
classifier = skflow.TensorFlowDNNClassifier(hidden_units=[70,80,70],
n_classes=5)
classifier.fit(X_train,y_train)
score = metrics.accuracy_score(y, classifier.predict(X))
print("Accuracy: %f" % score)
```

Performance Evaluation

The performance of the TensorFlowDeep neural network classifier is evaluated with predictive accuracy and training loss. 96.33% predictive accuracy is observed for the deep model built with nucleotide mapping. Also, its efficiency is compared against the linear classifier and the comparative results are tabulated in Table 7.1and shown in Fig.7.2, Fig.7.3, Fig.7.4.

Table 7.1 Predictive Performance of the Classifiers with Nucleotide Mapping

Model	Accuracy (%)	100 Iterations Average Training Loss	200 Iterations Average Training Loss
Linear Classifier	75	1.22664	1.02551
Deep Neural Network Classifier	96.33	1.20413	0.77192

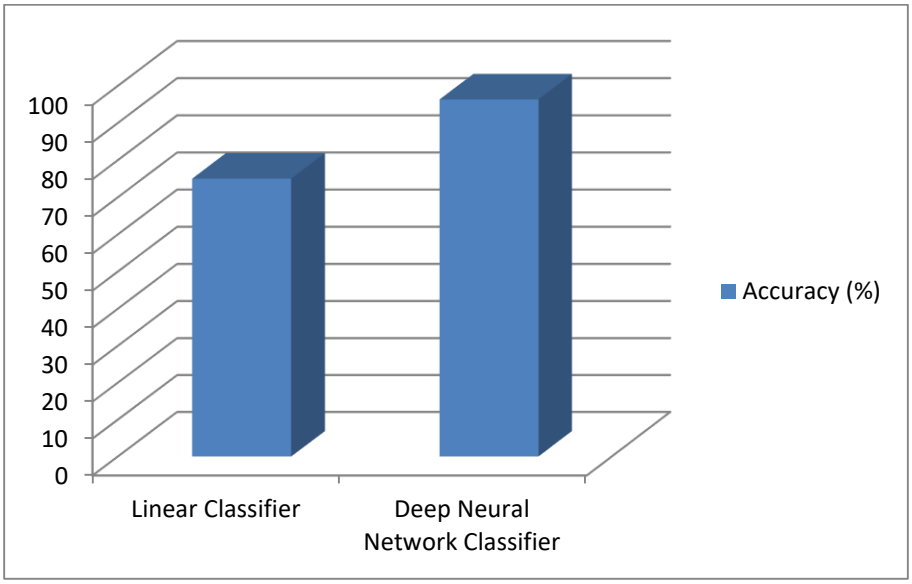

Fig.7.2 Performance of Classifiers with Tensorflow Deep Neural Network

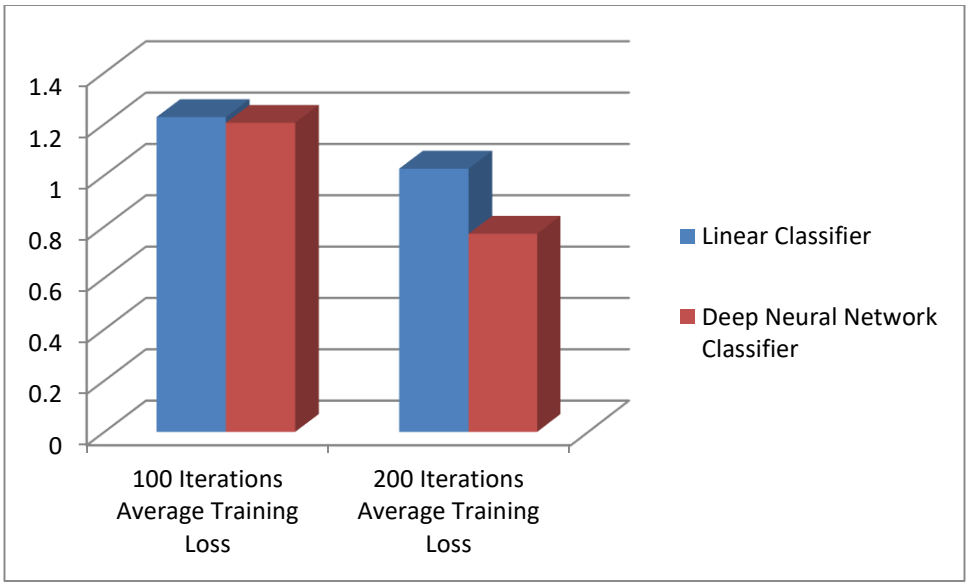

Fig.7.3 Performance of Classifiers in Average Training Loss with Tensorflow Deep Neural Network

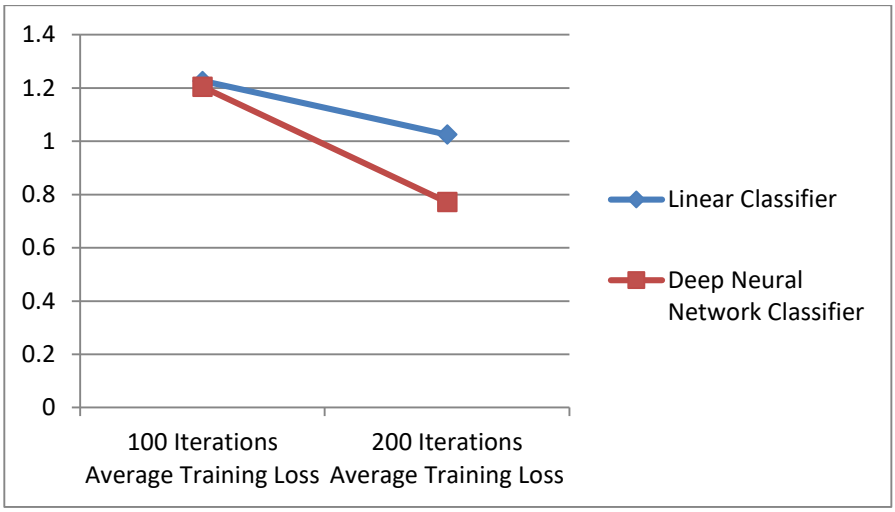

Fig.7.4 Average Training Loss of the Classifiers with Nucleotide Mapping

Findings

The deep neural network classifier outperforms well than linear classifier and gained an accuracy of 96.33% when nucleotide mapping scheme is adopted and the average loss also seem to be reduced.

7.3 Disease Identification Model using Deep Learning with Codon Mapping Scheme

The gene sequences code for a specific amino acid by translating it into its corresponding codons. A change in a codon reflects in protein folding or function which leads to disease. Hence, codon level mapping scheme is proposed in this experiment to built disease identification model using deep learning approach under Tensorflow environment.

Codon Mapping

The novel idea proposed in this work is to convert the DNA sequence into codon sequence with the codon-mapping scheme by splitting up the sequence into codons and the decimal values are bound into its matching 64 codons. Python script is written to translate the diseased gene sequences into its corresponding codon sequence. Converted decimal sequence forms an input array. This method converts array values into a sequence of decimal values for codons without losing position information of each nucleotide in sequences. Table 7.2 shows the codon and its corresponding integer identifiers.

Table 7.2 Codon and its Integer Identifiers

Codon(C)	Value(Vj)	Codon(C)	Value(Vj)	Codon(C)	Value(Vj)	Codon(C)	Value(Vj)
AAA	1	CAC	17	GAC	33	TAT	49
AAC	2	CAG	18	GAG	34	TCA	50
AAG	3	CAT	19	GAT	35	TCC	51
AAT	4	CCA	20	GCA	36	TCG	52
ACA	5	CCC	21	GCC	37	TCT	53
ACC	6	CCG	22	GCG	38	TGC	54
ACC	7	CCT	23	GCT	39	TGT	55
ACT	8	CGA	24	GGA	40	TTA	56
AGA	9	CGC	25	GGC	41	TTC	57
AGC	10	CGG	26	GGG	42	TTG	58
AGG	11	CGT	27	GGT	43	TTT	59
AGT	12	CTA	28	GTA	44	ATG	60
ATA	13	CTC	29	GTC	45	TGG	61

ATC	14	CTG	30	GTG	46	TGA	62
ATT	15	CTT	31	GTT	47	TAA	63
CAA	16	GAA	32	TAC	48	TAG	64

For example, consider the following gene sequence S

Triplet of size 3

S = CGTTAAATGCAAACGCTG

Codon

Gene sequence is converted into codon sequence as

S = CGTTAAATGCAAACGCTG

Mathematical modeling of codon mapping is stated below

$C(i) = V_j$

$C_m = a_i(C(i))$

Where C_m is Codon mapping array, a is an array of sequence, C - corresponding codon values and i = 1 to 64, j = 1 to 64]

Hence, Cm for S will be

$C_m(S) = [27, 63, 60, 16, 7, 30]$

Numpy array will be [27636016730]

Consider an example of diseased gene sequence,

>gi|383792147|ref|NM_170707.3| Homo sapiens lamin A/C (LMNA), transcript variant 1, mRNAATGGAGACCCCGTCCCAGCGGCGCGCCACCCGCAGCGGGGCGCAGGCCAGCTCCACTCCG CTGTCGCCCACCCGCATCACCCGGCTGCAGGAGAAGGAGGACCTGCAGGAGCTCAATGATCGCTT GGCGGTCTACATCGACCGTGTGCGCTCGCTGGAAACGGAGAACGCAGGGCTGCGCCTTCGCATCA CCGAGTCTGAAGAGGTGGTCAGCCGCGAGGTGTCCGGCATCAAGGCCGCCTACGAGGCCGAGCT CGGGGATGCCCGCAAGACCCTTGACTCAGTAGCCAAGGAGCGCGCCCGCCTGCAGCTGGAGCTG AGCAAAGTGCGTGAGTTTAAGGAGCTGAAAGCGCGCAATACCAAGAAGGAGGGTGACCTGATAG CTGCTCAGGCTCGGCTGAAGGACCTGGAGGCTCTGCTGAACTCCAAGGAGGCCGCACTGAGCACT GCTCTCAGTGAGAAGCGCACGCTGGAGGGCGAGCTGCATGATCTGCGGGGCCAGGTGGCCAAGC TTGAGGCAGCCCTAGGTGAGGCCAAGAAGCAACTTCAGGATGAGATGCTGCGGCGGGTGGATGC TGAGAACAGGCTGCAGACCATGAAGGAGGAACTGGACTTCCAGAAGAACATCTACAGTGAGGAG CTGCGTGAGACCAA

The equivalent codon Sequence is

ATGGAGACCCCGTCCCAGCGGCGCGCCACCCGCAGCGGGGCGCAGGCCAGCTCCACTCCGCTGTC
GCCCACCCGCATCACCCGGCTGCAGGAGAAGGAGGACCTGCAGGAGCTCAATGATCGCTTGGCGG
TCTACATCGACCGTGTGCGCTCGCTGGAAACGGAGAACGCAGGGCTGCGCCTTCGCATCACCGAGT
CTGAAGAGGTGGTCAGCCGCGAGGTGTCCGGCATCAAGGCCGCCTACGAGGCCGAGCTCGGGGAT
GCCCGCAAGACCCTTGACTCAGTAGCCAAGGAGCGCGCCCGCCTGCAGCTGGAGCTGAGCAAAGT
GCGTGAGTTTAAGGAGCTGAAAGCGCGCAATACCAAGAAGGAGGGTGACCTGATAGCTGCTCAGG
CTCGGCTGAAGGACCTGGAGGCTCTGCTGAACTCCAAGGAGGCCGCACTGAGCACTGCTCTCAGTG
AGAAGCGCACGCTGGAGGGCGAGCTGCATGATCTGCGGGGCCAGGTGGCCAAGCTTGAGGCAGCC
CTAGGTGAGGCCAAGAAGCAACTTCAGGATGAGATGCTGCGGCGGGTGGATGCTGAGAACAGGCT
GCAGACCATGAAGGAGGAACTGGACTTCCAGAAGAACATCTACAGTGAGGAGCTGCGTGAGACCA
AA

64 codon representation for a sequence is given below

124826241840565430265458643240305818252482022265410265684048444846840486414
554432143410465316542084728484231648545541026481747481614585448161862104430
303448304866445305444265461915304448543054840848858431653481444884332544126
444448614681129294029568444684829884218444830318582529657484454288486248837
4585662401630

Numerical representation of the given sequence is

124826241840565430265458643240305818252482022265410265684048444846840486414
554432143410465316542084728484231648545541026481747481614585448161862104430
303448304866445305444265461915304448543054840848858431653481444884332544126
444448614681129294029568444684829884218444830318582529657484454288486248837
4585662401630

Fig.7.5 Translating Codon Sequence into Numerical Array

Building the Model

To build the deep model, the 1D arrays formed using Codon mapping scheme are further converted into numpy arrays. The data frame is built with the numpy array. Appropriate hyper-parameters such as the number of units per layer, learning rate and the epochs of iteration are chosen similar to first experiment (refer section 7.2) for learning the deep neural network. Here,

Number of layers = 3

Hidden layers = 70, 80, 70

Drop out rate = 0.5

Epochs of iteration = 100

The script for building the classifier is shown below.

Script for TensorflowDeepNeuralNetworkClassifier

```
import tensorflow as tf
import numpy as np
from numpy import genfromtxt

# Load datasets
my_data = genfromtxt('codon.csv', delimiter=',')
from sklearn.cross_validation import train_test_split
X = my_data[:,-1]
y = my_data[:,1]
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
from sklearn import metrics

# Build 3 layer DNN with 70, 80, 70 units respectively.
classifier = skflow.TensorFlowDNNClassifier(hidden_units=[70,80,70], n_classes=5)

# Fit model.
classifier.fit(X_train,y_train)
score = metrics.accuracy_score(y, classifier.predict(X))

# Evaluate accuracy.
print("Accuracy: %f" % score)
```

Performance Evaluation

The performance of the TensorFlowDeep neural network classifier based on codon mapping is evaluated with predictive accuracy and training loss. 96% predictive accuracy is observed for the deep model built with nucleotide mapping. In addition, its efficiency is compared against the linear classifier and the comparative results are tabulated in Table 7.3 and shown in Fig.7.6, Fig.7.7, Fig.7.8.

Table 7.3 Predictive Performance of the DNN with Codon Mapping

Model	Accuracy (%)	100 Iterations Average Training Loss	200 Iterations Average Training Loss
Linear Classifier	70	1.23040	1.04786
Deep Neural Network Classifier	96	1.11547	0.46410

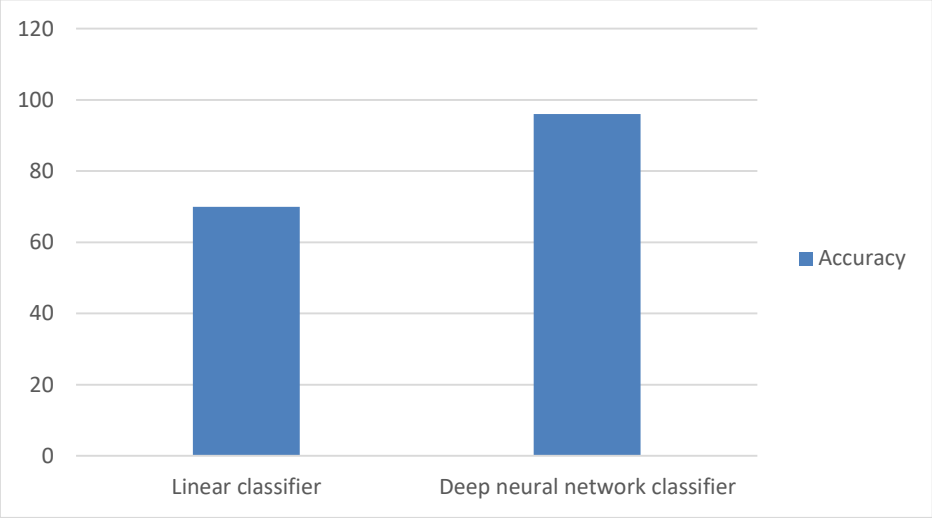

Fig.7.6 Performance of Deep net using Codon Mapping

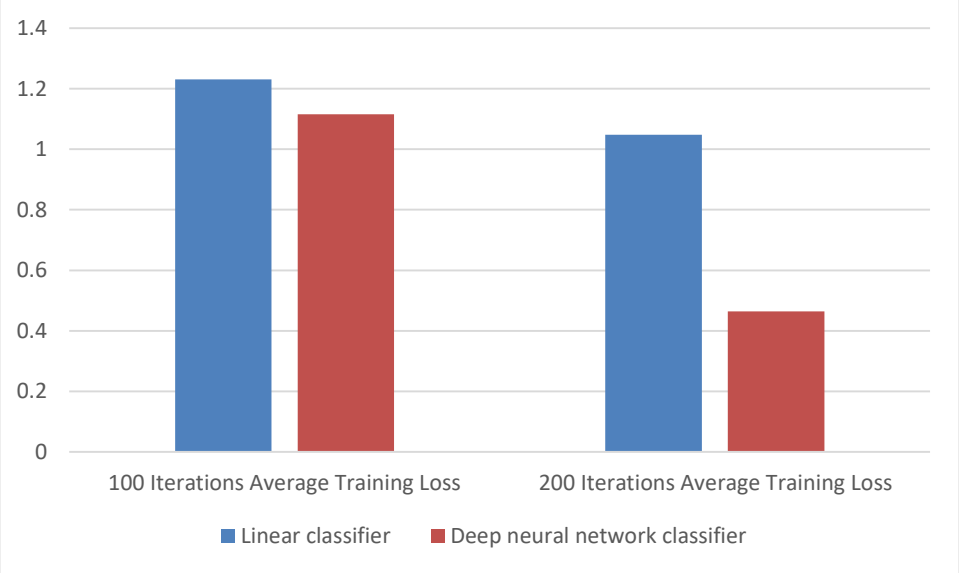

Fig.7.7 Performance of DNN with Average Training loss using Codon Mapping

Fig.7.8 Average Training loss of the Classifiers with Codon Mapping

Findings

In this research, improvements are achieved by using deep neural network, with the high power of representing a complicated problem like disease identification from gene sequences. Two kind of mapping schemes are used to represent sequences, so that the specific position information of each individual nucleotide in sequences can be preserved. The proposed approach can learn complex and abstract features at higher layers representation by greedy layer-wise training auto-encoders with large amount of data. Through the results of several experiments, it was recognized that the hyper-parameters significantly improves the prediction performance of the model.

Classifying the type of diseases by transforming gene sequences into numeric array based on mapping techniques is never attempted in literature. It is confirmed that Deep Neural Network in tensor flow environment with the hidden layer architecture extract features on their own and achieved an intensified results for predicting the disease from the mutated gene sequences. It was also found that the appropriate increase in depth of the network can stimulate the performance of DNN. Parallelization is achieved with the tensor flow environment that aids in improving the performance of the CPU in windows environment. The experimental results show that DNN approach can achieve better performance than linear classifier.

7.4 Comparison of Models based on ANN and DNN

To enhance the performance of the shallow learning model, deep architectures are designed with additional hyper parameter such as the learning rate of the auto encoder. The main difference in training the DNN and NN is the parameter initialization. Only small weights are assigned in NN whereas the DNN weights are generated pretraining to autoencoder. Also the number of network layers is limited to 3 in NN and the performance degrades when the depth of the layer increases. With the increase of the network layers in the shallow architecture the performance of NN gradually reduces. The performance of DNN increases when the network layer increases from two to four. When the network layer increases to five, the performance remains stable. 3-NN denotes the NN with three layers and 3-DNN denotes the three layered DNN, and so on. The increase of the network depth will helps in increasing the network capacity.

The performance of the disease identification models built using Deep neural network with two kinds of mapping schemes is compared with the results of supervised disease identification model built using Artificial Neural Network (refer Chapter 5) and the performances are analysed with respect to various measures. The performance comparison between shallow net and deep net are depicted in Table 7.3, Table 7.4, Table 7.5 and Fig.7.9, Fig.7.10, Fig.7.11, Fig.7.12.

In nearly all cases, DNN performs better when the network layer increases from three to four. A speculative inspiration for deep architectures denotes that deep architectures were much more efficient than shallow architectures, in terms of computational components that are required to characterize certain functions. Therefore, in this work depth of DNN is tested from varying the number of hidden layers from three to eight.

Table 7.4 Predictive Performance of the ANN Classifier

Performance criteria	**Artificial Neural Network**
Correctly classified Instance	829
Incorrectly classified instance	171
Prediction accuracy	**82.9%**
Precision	0.829
Recall	0.82
F1 Score	82.1
Cohen’s Kappa	0.83
Time taken to build the model (in sec)	9.7

Table 7.5 Performance Comparison of Shallow Net with Deep Net (with hidden layers ranging from 3 to 8 in Nucleotide Mapping Scheme)

Performance Criteria	2-NN	3-NN	3- DNN	4-DNN	5-DNN	6-DNN	7- DNN	8-DNN
Precision	71.43	72.14	79.7	86.7	84.52	83.5	81.98	80.23
Recall	80.2	82	89.4	91.89	90.1	89.7	87.98	85.98
F-Score	75.56	66.86	84.3	89.21	87.22	85.99	84.7	83.98
Accuracy	82.1	79.3	83.4	92.33	91.3	90.7	89.8	88.7

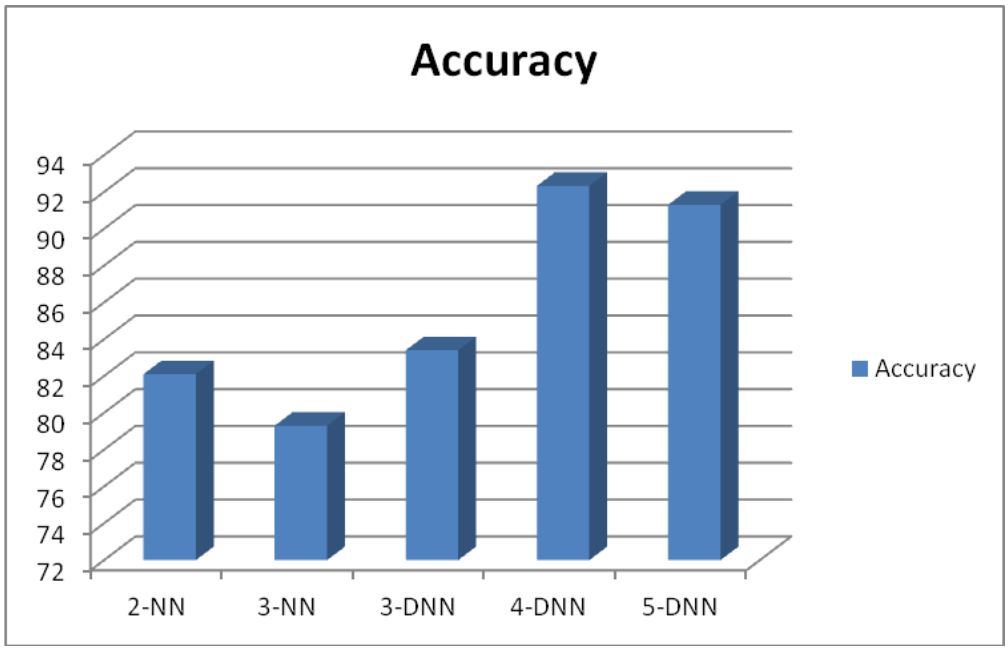

Fig.7.9 Comparison on Predictive Accuracy Nucleotide Mapping

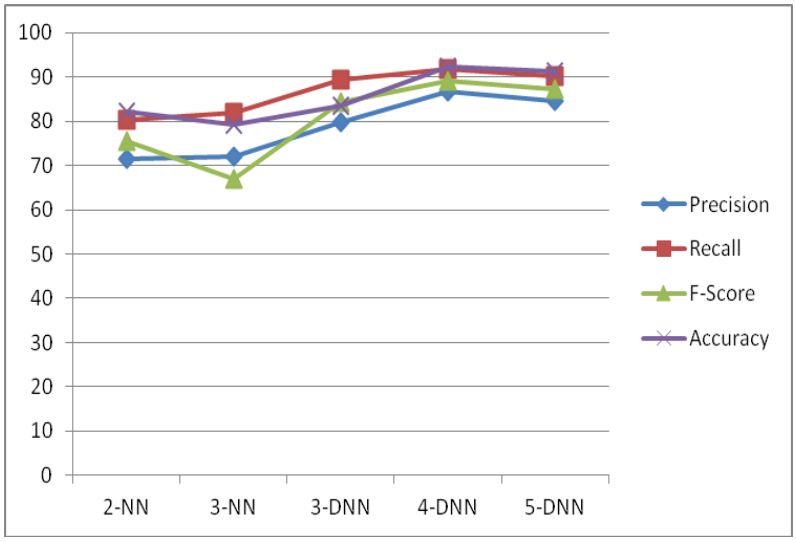

Fig.7.10 Performance of Evaluation Measures in Nucleotide Mapping

Table 7.6 Performance Comparison of Shallow Net with Deep Net (with hidden layers ranging from 3 to 8 in Codon Mapping Scheme)

Performance Criteria	**2 -NN**	**3- NN**	**3-DNN**	**4-DNN**	**5-DNN**	**6-DNN**	**7-DNN**	**8-DNN**
Precision	71.43	72.14	80.7	85.3	83.12	82.15	81.54	80.37
Recall	80.2	82	90.4	91.2	89.78	88.17	87.28	86.84
F-Score	75.56	66.86	85.3	88.15	87.32	86.49	85.37	84.18
Accuracy	82.1	79.3	86.8	91	90.84	89.17	88.58	87.67

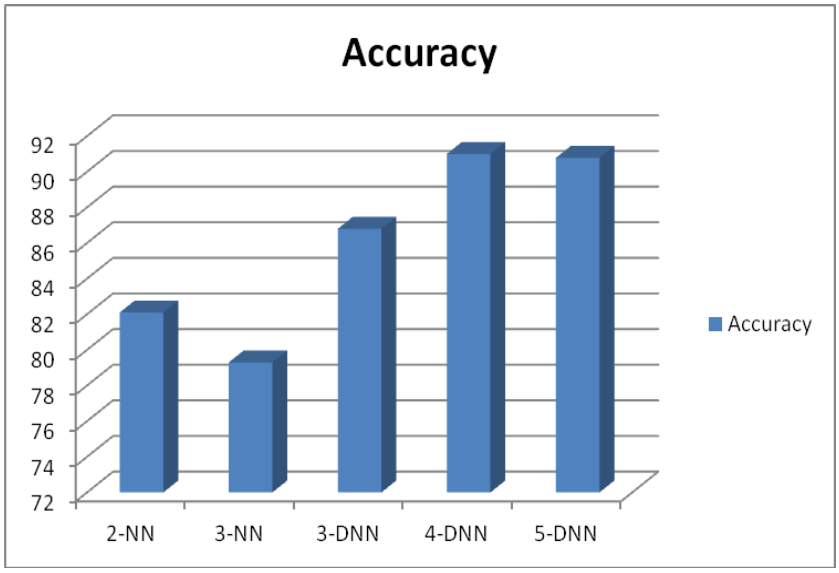

Fig.7.11 Comparison on Predictive Accuracy (Codon mapping)

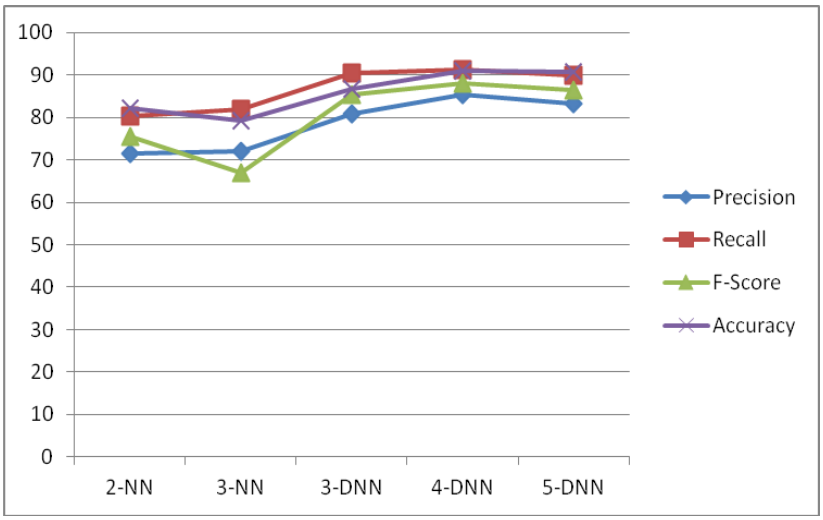

Fig.7.12 Performance of Evaluation Measures in Codon Mapping

Findings

Improvements are achieved by using deep neural network, with the high power of representing a complicated problem like disease identification from gene sequences. Two kind of mapping schemes proposed here aids in identifying the mutations occurred in the gene sequence by retrieving the specific position information of each individual nucleotide and thereby enabling accurate disease identification. The proposed approach can learn complex and abstract features at

higher layers representation by greedy layer-wise training auto-encoders with large amount of unlabeled data. Through the results of these experiments, it was observed that the proper selection of hyper-parameters significantly improves the prediction performance of the models. It is confirmed that Deep neural network in tensor flow environment with the hidden layer architecture extract features on their own and achieved an intensified results for predicting the disease from the mutated gene sequences.

7.5 Summary

This chapter portrayed the modeling of muscular dystrophy disease identification model through deep neural network. Diseased gene sequences have been encoded into decimal format through nucleotide and codon mapping schemes. Self extraction of features from the gene sequences is done through stacked auto encoder. Models were built using deep neural network in tensorflow environment. Tensorflow deep neural network classifier facilitates a better improvement in predictive accuracy when compared with Artificial Neural Network.

Remarks

1. Paper titled, "Nucleotide and codon mapping schemes for deep learning to diagnose muscular dystrophy", has been accepted for publication in Frontiers in Biosciences (**SCIE Indexed**)