Pattern Recognition and Neural Networks

Assignment – 3

Name – Saankhya Subrata Mondal Sr. No. – 17990

Dept. – CSA Degree – M. Tech in AI

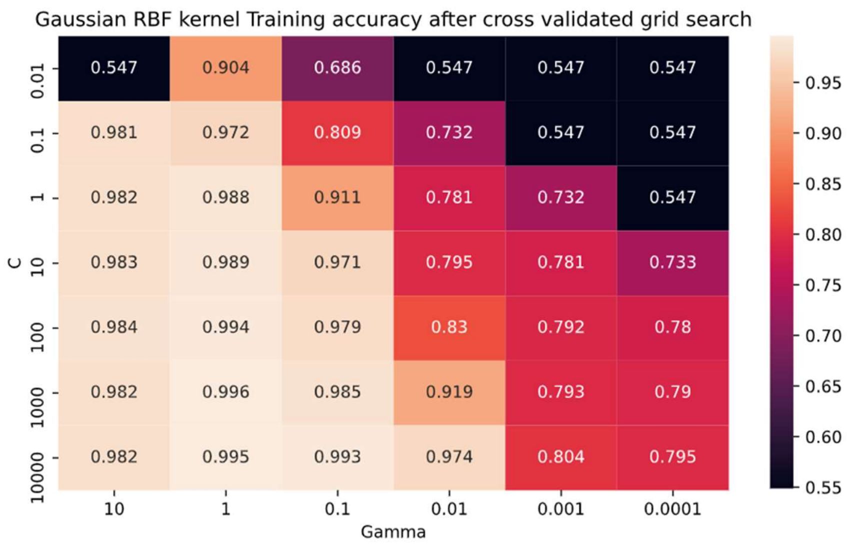
For all the problems, the best obtained classifier model has been discussed in detail. Training and testing (validation in case of neural network models) accuracy have been provided. The model accuracy and loss graphs have been plotted (in case of neural networks). The test set confusion matrices have been shown to have a better look at how the classifier has performed on the test data. Precision, Recall, and F1 score metrics have been discussed whenever felt necessary. Additionally, various neural networks models were explored differently in different problems. These include performance analysis on a given problem and dataset by varying hyper-parameters like learning rate, number of hidden layers, choice of optimizers, choice of activation function, or regularization techniques. In case of SVMs, performance analysis using various kernels, and choice of hyper-parameters has been done. Tolerance has been set to 0.001. Hyper-parameter tuning has been done using cross-validated grid search over the parameters (with a given range of each) using various subsets of the training data. Using the best combination of hyper-parameters for each sub-problem, the performance over the test set has been observed. Note that gamma and degree hyper-parameters are not used in case of linear kernel. Degree parameter is used only in case of polynomial kernel. Finally for the given problem, comparison study of performance between all the classifiers have been conducted. For SVMs, I have used sklearn library and for NNs, I have used Tensorflow library.

Question 1

The problem is 2-class classification. The data here is 2-D and there are 2000 samples in total. There are three data files with different label noise levels. As a part of data preprocessing, all the features were normalized.

Classification using Support Vector Machines –

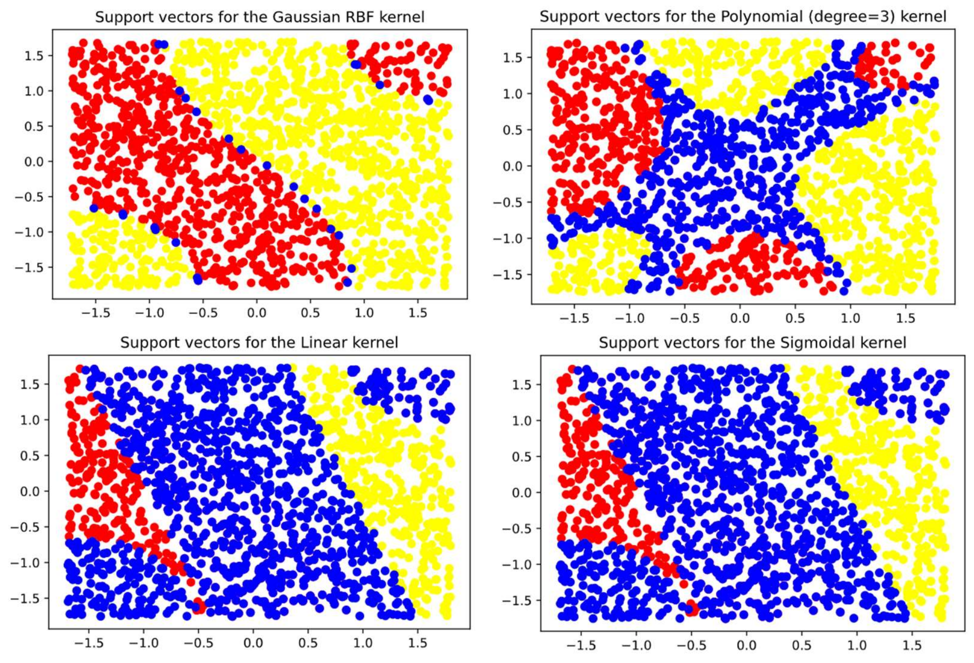
All the four kernels, Gaussian RBF, polynomial, linear, and sigmoidal have been explored. For illustration purposes, a heatmap of the hyper-parameter grid search has been shown for the Gaussian RBF kernel SVM trained on 'data_noise_0.txt'. As it can be observed, the



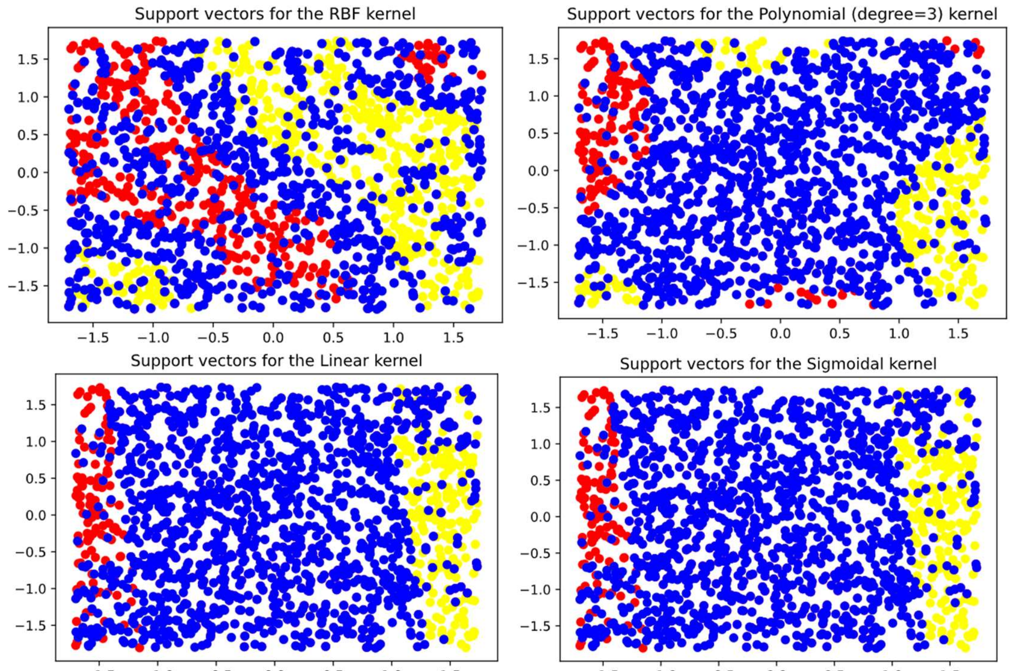
training accuracy obtained by using parameters C = 1000 and gamma = 1 is the highest. Hence, it is the best fit model for this case. Similarly, for all other cases, such heatmaps were examined.

In the following plots, the support vectors (in blue) are shown in a scatter plot for each kernel in the feature normalized space.

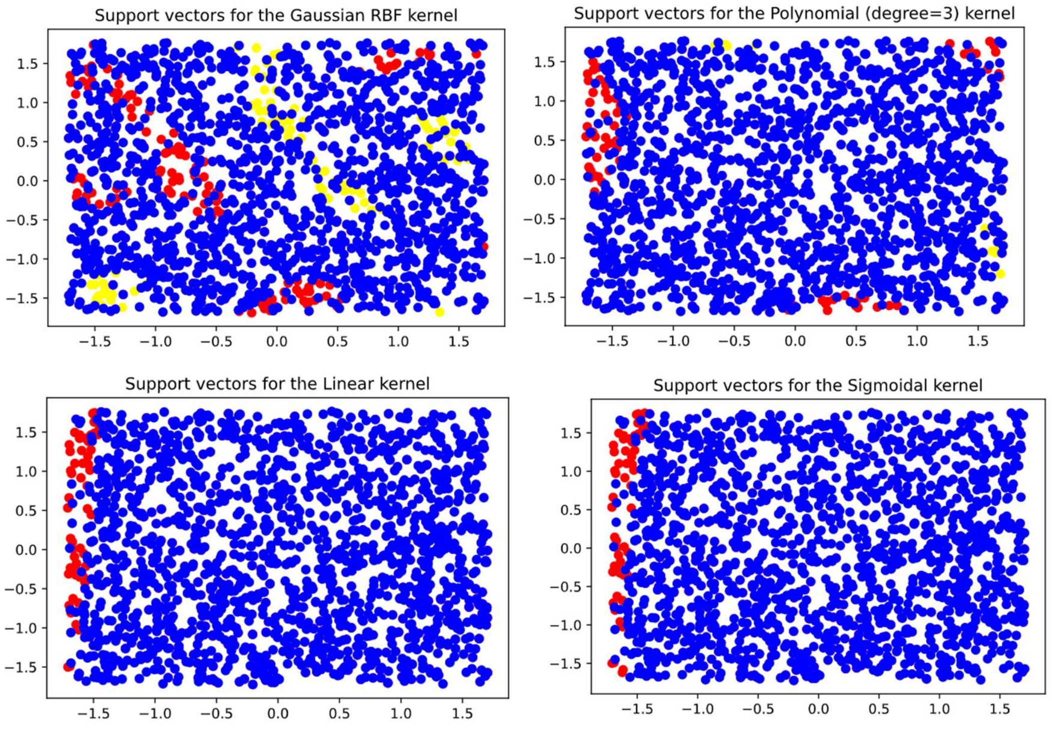
a) data_noise_0.txt



b) data_noise_20.txt



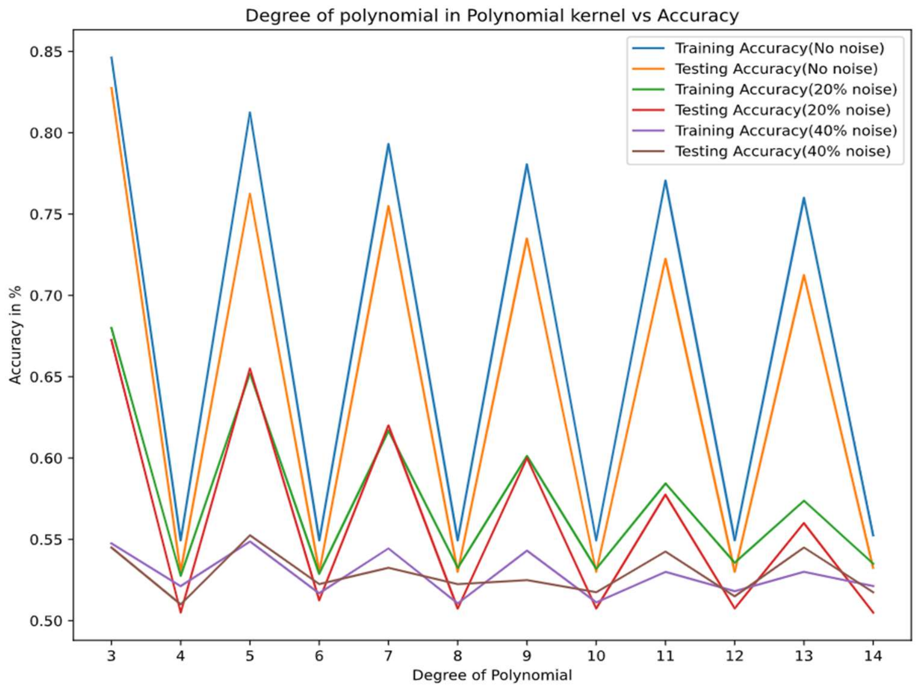
c) data_noise_40.txt



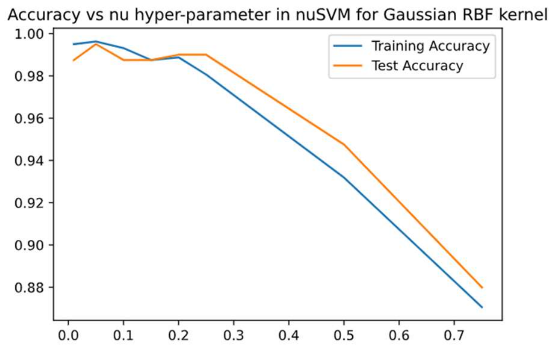
The results for the best obtained model after grid search (over hyper-parameter C and gamma) are tabulated.

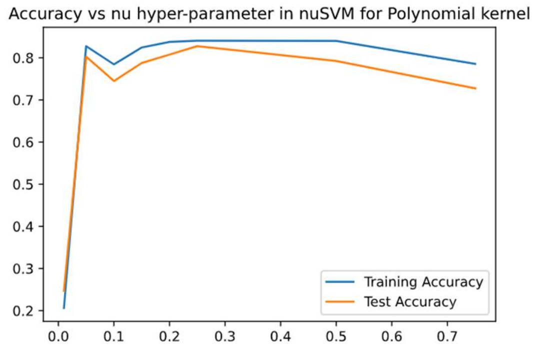
Dataset	Kernel	Best fit hyper-parameters	Number of Support Vectors	Training Accuracy	Test Accuracy
data_noise_0.txt	Gaussian RBF	C = 1000, Gamma = 1	14, 14	99.6%	99.25%
data_noise_0.txt	Polynomial	C = 1 Gamma = 1 Degree = 3	335, 336	84.44%	86%
data_noise_0.txt	Sigmoid	C = 1000 Gamma = 0.001	543, 543	77.6%	80.5%
data_noise_0.txt	Linear	C = 10	525, 524	79.38%	77.5%
data_noise_20.txt	Gaussian RBF	C = 1000, Gamma = 1	391, 395	79.4%	74.5%
data_noise_20.txt	Polynomial	C = 0.01 Gamma = 10 Degree = 3	614, 614	67.7%	65.5%
data_noise_20.txt	Sigmoid	C = 1000 Gamma = 0.001	659, 658	64.1%	66.25%
data_noise_20.txt	Linear	C = 1	652, 652	65.2%	65.5%
data_noise_40.txt	Gaussian RBF	C = 10, Gamma = 1	699, 698	58.3%	55.75%
data_noise_40.txt	Polynomial	C = 1 Gamma = 10 Degree = 3	759, 760	53.2%	56.5%
data_noise_40.txt	Sigmoid	C = 0.01 Gamma = 0.1	776, 773	51.7%	52.25%
data_noise_40.txt	Linear	C = 0.01	774, 773	51.7%	52.14%

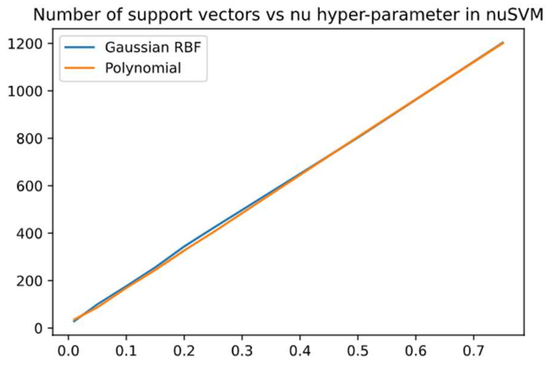
The following plot shows the training and testing accuracies of all the three datasets using Polynomial kernels with different degree of polynomial.



Let us now use nu-SVM for the first dataset using Gaussian RBF and polynomial kernel. Let us vary the parameter nu and observe the performance and number of support vectors.







Observations –

- The Gaussian RBF kernel gives the best results with and without label noise. The support vectors (the blue scatter points in scatter plots) are very less in number for all the cases for Gaussian RBF kernel. This is because we are not allowing much slack (allowing small margin) and still, we are getting good accuracy. The Gaussian RBF kernel seems to perform well even in presence of label noise.
- From the grid search heatmap, for every value of gamma, the training accuracy always increases with increasing value of C.
- The polynomial kernel (cubic) gives second best results. However, such results are obtained only after reducing the hyper-parameter C that is by using a larger margin. The number of support vectors are larger in this case due to the larger amount of slackness allowed.
- From the plot, it can be observed that odd degree polynomial kernels perform better than all even degree polynomial kernels for all the three datasets. The accuracies though reduce with increase in degree of even/odd degree polynomial kernels.
- Linear and Sigmoid kernels perform similarly. Almost all the training data are used as support vectors. This indicates they are not a good kernel for classification in this case.
- In nu-SVMs, it was observed that by controlling the nu parameter, we are controlling the number of support vectors to be used. Lower value of nu implies lesser number of support vectors. Similar monotonically increasing relation was observed in the other two datasets. Also, Gaussian RBF kernel performs better than the polynomial kernel.

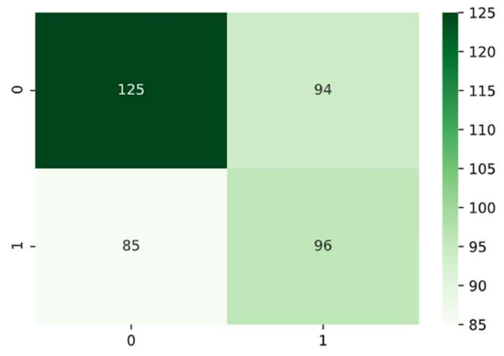
Classification using feedforward neural networks –

The dataset was split into three parts, 72% for training, 8% for cross validation, and 20% for testing. Binary cross entropy loss was used as objective function for minimization. Adam optimizer with learning rate 0.1 and minibatch size 32 was used. The learning rate was reduced by a factor of 0.1 every time same model loss was obtained for 5 consecutive epochs. ReLU activation function was used. No regularization was performed at first. The training data was shuffled after every epoch. A feedforward neural network with one hidden layer containing 8 neurons was used first for all the three cases.

Dataset	Training Accuracy	Validation Accuracy	Testing Accuracy
data_noise_0.txt	99.58%	99.75%	99.5%
data_noise_20.txt	77.54%	74.50%	72.5%
data__noise_40.txt	55.62%	52.50%	49%

In part b), the model seems to perform well, given that there is 20% label noise. However, the model is slightly overfitting due to poorer validation and test results. After applying L2 – regularization with parameter 0.0001, a training accuracy of 78.8% is obtained. Validation accuracy is an excellent 80% whereas test accuracy improves to 77.8%.

In part c), the results are unsatisfactory. In fact, the model underfits. Let us use a complex and deeper architecture to get better results. Now, the neural network has two hidden layers with 8 and 16 neurons, respectively. The network was trained using minibatch gradient descent. A training accuracy of 57.89% was obtained which is satisfactory since there is 40% label noise. The validation accuracy obtained is 58.13% and test set accuracy was 55.25%. The test set confusion matrix has been shown. The table summarizes the performance for the best model obtained in each case.



Dataset	Training Accuracy	Validation Accuracy	Testing Accuracy
data_noise_0.txt	99.58%	99.75%	99.5%
data_noise_20.txt	78.8%	80%	77.8%
data__noise_40.txt	57.89%	58.13%	55.25%

Observations -

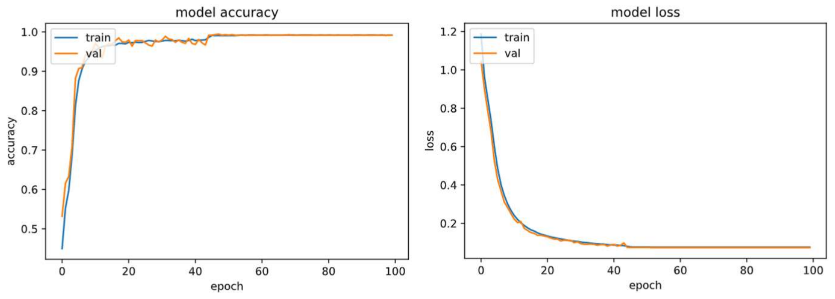
- Neural networks with proper hyper-parameter tuning performed better than SVMs with Gaussian RBF kernel in the cases where 20% and 40% label noise were present. SVMs were more sensitive to label noise that neural networks.
- With no label noise, both neural network and SVM with Gaussian RBF kernel performed excellently attaining 99+% accuracies on training and test sets.
- In neural networks, it was observed how adding a slight L2 regularization term was enough to mitigate overfitting.

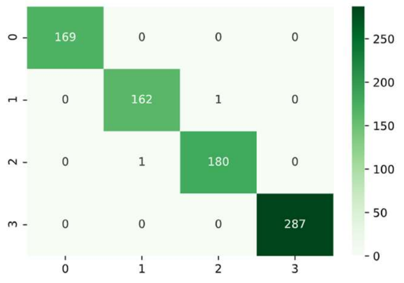
Question 2

The problem is 4-class classification. The data here is 2-dimensional and there are a total of 8000 samples. There are three different data files with different label noise levels. As a part of data preprocessing, all the features were normalized, and the output labels were encoded into a 'one hot vector' representation.

Classification using feedforward neural networks –

First, classification using feedforward neural network has been explored. The dataset was split into three parts, 80% for training, 10% for cross validation, and 10% for testing. Categorical cross entropy loss was used as objective function for minimization. RMSprop with learning rate 0.1 and minibatch size 32 was used. The learning rate was reduced by a factor of 0.1 every time same model loss was obtained for 5 consecutive epochs. ReLU activation function was used. No regularization was performed at first. The training data was shuffled after every epoch. A feedforward neural network with one hidden layer containing 8 neurons was used first for all the three cases. Following are the model accuracy, and model loss curves, and the confusion matrix for the board_data.txt.



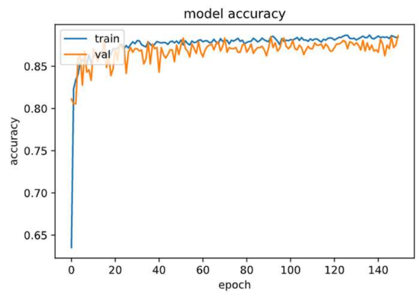


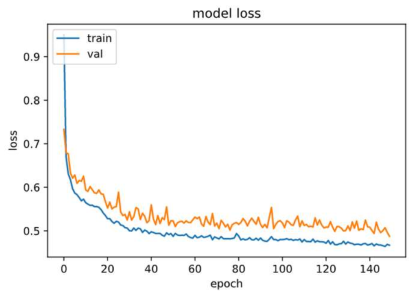
This table shows the accuracy of the above model on all the three datasets.

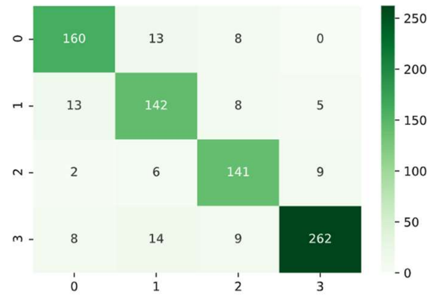
Dataset	Training Accuracy	Validation Accuracy	Testing Accuracy
board_data.txt	99.81%	99.86%	99.75%
board_data_10.txt	86.11%	88.33%	86.625%
board_data_25.txt	64.98%	65.28%	64.875%

The accuracy obtained in the latter two cases are not satisfactory. Hence, a more complex neural network is needed to be trained.

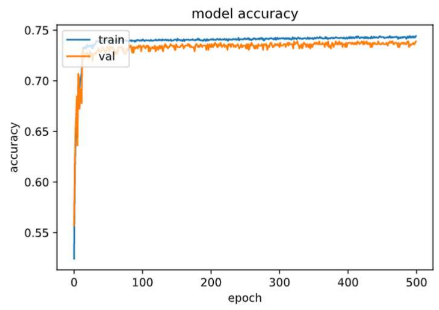
For part b) (with 10% label noise), a neural network consisting of two hidden layers with 16 and 8 neurons was trained. No regularization was performed. Adam optimization algorithm was used. An accuracy of 89% was obtained on the training set. Validation accuracy was 88.61% and test accuracy obtained was 88.125%. This model performs better than the previous one. The model accuracies are appreciable since 10% of the data is labelled wrongly. Following are the model accuracy, and model loss curves, and the confusion matrix for the board_data_10.txt.

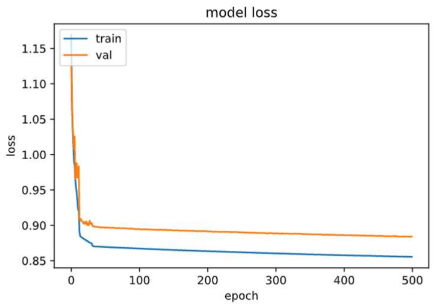


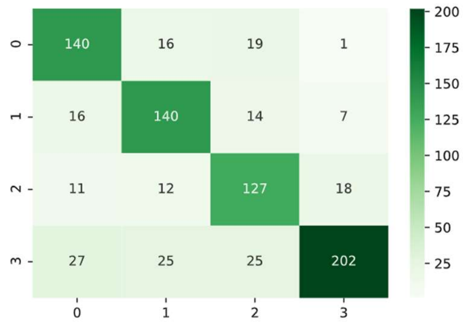




For part c) (with 25% label noise), a neural network consisting of three hidden layers with 16, 32, and 16 neurons each was used to train this dataset. Training was done for 500 epochs. Minibatch gradient descent was used with initial learning rate of 0.1. The learning rate was reduced by a factor of 0.1 every time same model loss was obtained for 5 consecutive epochs. An accuracy of 74.28% was obtained on the training set. Validation accuracy was 73.89% and test accuracy obtained was 76.125%. This model performs better than the previous one. The model accuracies are appreciable since 25% of the data is labelled wrongly. Following are the model accuracy, and model loss curves, and the confusion matrix for the board_data_25.txt.



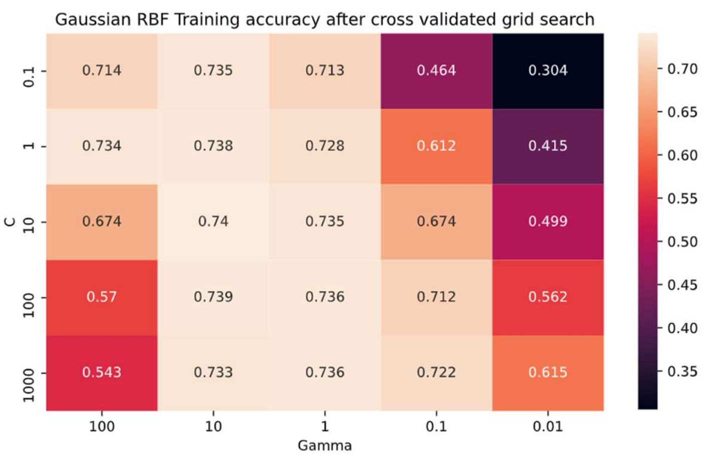




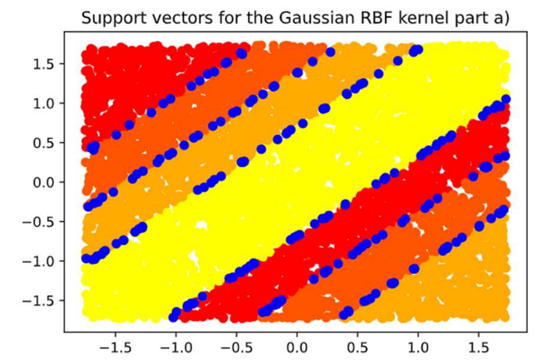
The following table summarizes the performance for the best model obtained in each case.

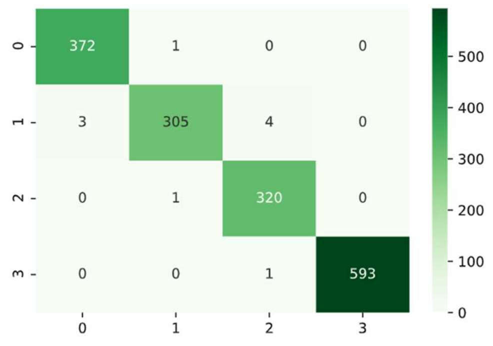
Dataset	Training Accuracy	Validation Accuracy	Testing Accuracy
board_data.txt	99.81%	99.86%	99.75%
board_data_10.txt	89%	88.61%	86.125%
board_data_25.txt	74.28%	73.89%	76.125%

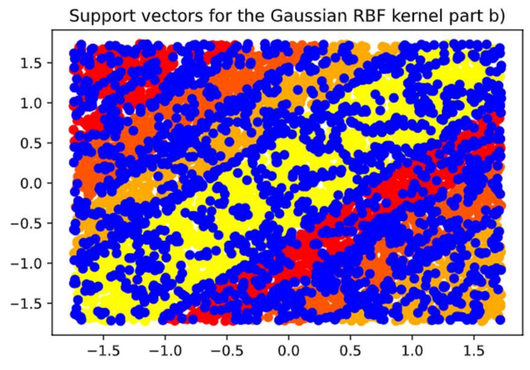
Classification using Support Vector Machines –

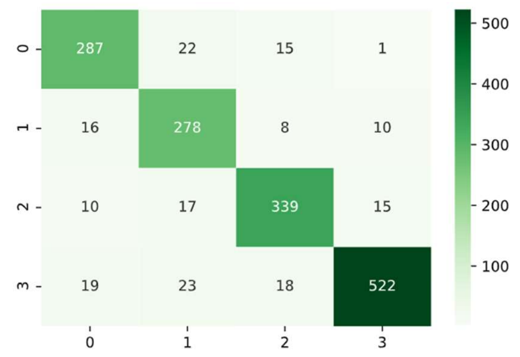


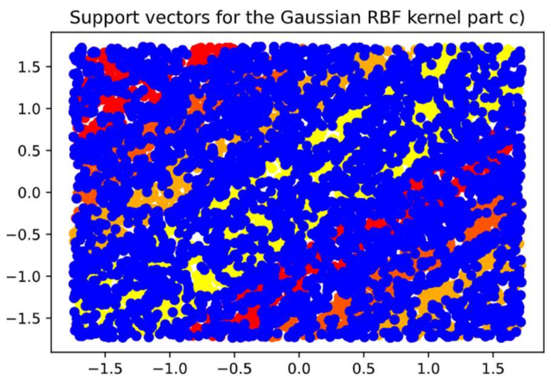
Here, the dataset has been split into training and test sets with an 80%-20% split. The training and test set performance of the best fit models in each case along with their hyper-parameters and number of support vectors are tabulated. Tolerance parameter has been set to 0.001. For illustration purposes, a heatmap of the hyper-parameter grid search has been shown for the Gaussian RBF kernel SVM trained on 'board_data_25.txt'. As it can be observed, the training accuracy obtained by using parameters C = 10 and gamma = 10 is the highest. Hence, it is the best fit model for this case. The support vectors (in blue) are shown in a scatter plot for Gaussian RBF kernel in the feature normalized space for each dataset. Confusion matrices have also been shown for each part.

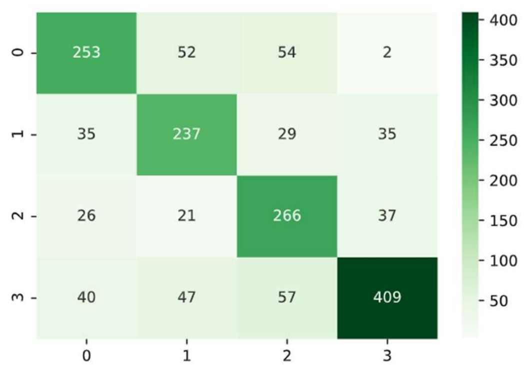












Dataset	Kernel	Best fit hyper-parameters	Number of Support Vectors	Training Accuracy	Test Accuracy
board_data.txt	Gaussian RBF	C = 1000, gamma = 1	43, 49, 41, 39	99.4%	99.32%
board_data.txt	Sigmoid	C = 100, Gamma = 0.01	1398, 1325, 1399, 2278	38.5%	43.69%
board_data.txt	Linear	C = 1	1400, 1294, 1420, 2196	38.57%	34.93%
board_data.txt	Polynomial	C = 10, Gamma = 1 Degree = 6	1382, 861, 1380, 196	73.01%	73.81%
board_data_10.txt	Gaussian RBF	C = 100, Gamma = 10	474, 535, 528, 520	89.09%	89.125%
board_data_10.txt	Sigmoid	C = 10, Gamma = 0.01	1402, 1389, 1474, 2134	33.4%	34%
board_data_10.txt	Linear	C = 0.1	1426, 1390, 1484, 1922	33.4%	36.1%
board_data_10.txt	Polynomial	C = 10, Gamma = 0.1 Degree = 4	1413, 1308, 1473, 1019	55.54%	55.38%
board_data_25.txt	Gaussian RBF	C = 10, Gamma = 10	921, 910, 987, 904	74%	73%
board_data_25.txt	Sigmoid	C = 100, Gamma = 0.01	1461, 1426, 1569, 1944	32.7%	29.31%
board_data_25.txt	Linear	C = 1	1428, 1388, 1448, 1912	32.1%	32.8%
board_data_25.txt	Polynomial	C = 10, Gamma = 0.1 Degree = 4	1475, 1421, 1567, 1451	37.89%	37.10%

Observations -

- Neural networks with proper hyper-parameter tuning performed better than SVMs with Gaussian RBF kernel in the cases where 10% and 25% label noise were present. SVMs were more sensitive to label noise that neural networks.
- With no label noise, both neural network and SVM with Gaussian RBF kernel performed remarkably well.

- Model accuracy and loss curves were used extensively to check whether the NN model was overfitting or not.
- Various activation functions like ReLU, sigmoid, tanh, and leaky ReLU were tried. They produced similar results for this problem.
- In case of SVMs, Gaussian kernel performed way better than any other kernel. Polynomial (cubic) kernel was the second-best.

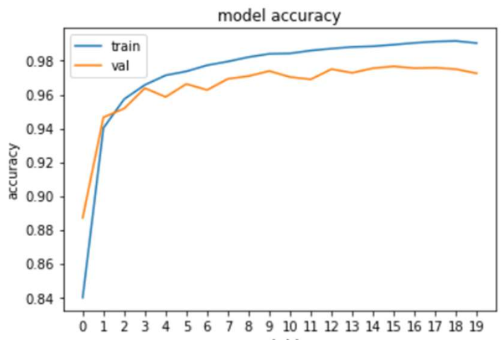
Question 3

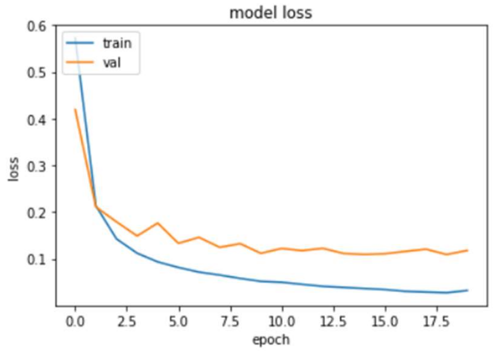
The training dataset consists of 50000 images. The test dataset consists of 10000 images. All images are of shape 28x28. The data stored in '.mat' format is a 784-length vector. This is a 10-class classification problem.

MNIST Dataset using CNN –

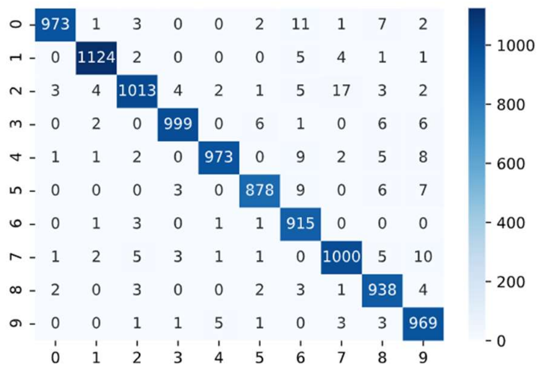
The data was reshaped into 2-D features. The training data was further divided to facilitate cross validation. 10% of the training data was used for validation. Categorical cross entropy loss was used as objective function. The CNN model architecture has been shown in the following table.

Layer	Hyper-parameters	Output shape
Conv2D	Filters – 8, Shape – 3x3, Strides – 1, Padding - done	28x28x8
MaxPool2D	Shape – 2x2, Strides – 1	14x14x8
Conv2D	Filters – 16, Shape – 3x3, Strides – 1, Padding - done	14x14x16
MaxPool2D	Shape – 2x2, Strides – 1	7x7x16
Conv2D	Filters – 32, Shape – 3x3, Strides – 1, Padding - done	7x7x32
MaxPool2D	Shape – 2x2, Strides – 1	3x3x32
Fully Connected Layer		288
Output		10

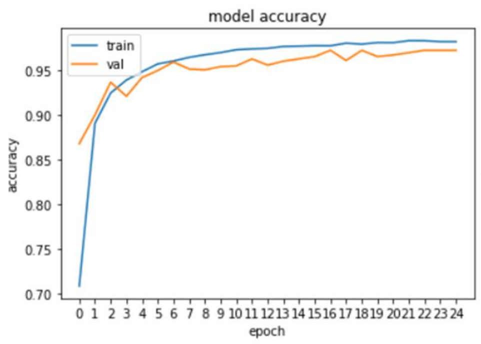


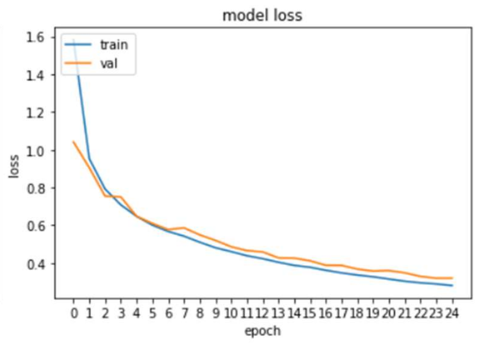


'ReLU' activation function has been used except in the last layer where 'softmax' function is used. The filter weights are initialized using 'He' initialization method. No regularization technique has been used initially. There are a total of 8778 parameters used. The model has been trained for 20 epochs. The training data was shuffled after every epoch. Minibatch gradient descent algorithm ('sgd' in Tensorflow) with learning rate 0.01 and batch size 32 was used for optimization. Following are the 'Accuracy vs Epoch' and 'Loss vs Epoch' curve. Train accuracy – 99.15%, Validation accuracy – 97.26%, Test accuracy – 97.91%. The confusion matrix obtained from the test set is shown. Let us observe the above two curves. The model loss is higher in the last few epochs. There is a possibility of overfitting. Hence, let us apply regularization techniques to eliminate overfitting. We can use 'L2 regularization' and 'Dropout'. I have used 'L2 regularization' to see the effect of regularization. The L2 parameter used was 0.005.

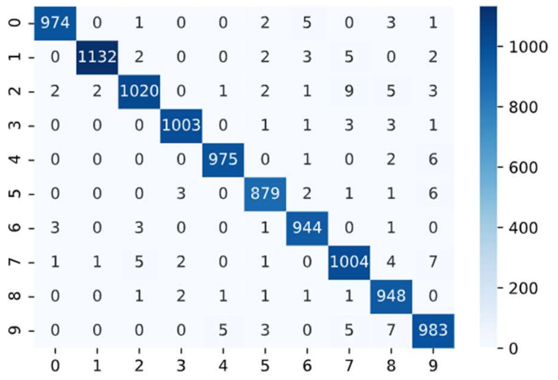


	0	1	2	3	4	5	6	7	8	9
0	973	1	3	0	0	2	11	1	7	2
1	0	1124	2	0	0	0	5	4	1	1
2	3	4	1013	4	2	1	5	17	3	2
3	0	2	0	999	0	6	1	0	6	6
4	1	1	2	0	973	0	9	2	5	8
5	0	0	0	3	0	878	9	0	6	7
6	0	1	3	0	1	1	915	0	0	0
7	1	2	5	3	1	1	0	1000	5	10
8	2	0	3	0	0	2	3	1	938	4
9	0	0	1	1	5	1	0	3	3	969





Now, the difference between the model losses is very less as the validation curve follows the training curve. Now, it can be concluded that the model does not overfit. Here, Train accuracy – 99.17%, Validation accuracy – 98.52%, Test accuracy – 98.61%. This model even performs better than the previous model. The confusion matrix obtained from the test set is as shown on the right.



	0	1	2	3	4	5	6	7	8	9
0	974	0	1	0	0	2	5	0	3	1
1	0	1132	2	0	0	2	3	5	0	2
2	2	2	1020	0	1	2	1	9	5	3
3	0	0	0	1003	0	1	1	3	3	1
4	0	0	0	0	975	0	1	0	2	6
5	0	0	0	3	0	879	2	1	1	6
6	3	0	3	0	0	1	944	0	1	0
7	1	1	5	2	0	1	0	1004	4	7
8	0	0	1	2	1	1	1	1	948	0
9	0	0	0	0	5	3	0	5	7	983

Let us now observe the performance on the full test set by using training set containing 5000, 10000, and 25000 training samples.

Training set size	Train Accuracy	Validation Accuracy	Test Accuracy	Observation
5000	97.51%	96.04%	95.27%	The model with above set L2 parameter was overfitting. Hence, L2 parameter was set to 0.02 and the model was trained for 100 epochs.
10000	97.57%	97.00%	97.00%	The model was trained with L2 parameter 0.01 and reaches convergence in 65 epochs.
25000	97.67%	97.70%	97.03%	The model was trained with L2 parameter 0.01 and performs like the previous one.
50000	99.17%	98.52%	98.61%	The model was trained with L2 parameter 0.005 and reaches convergence in 25 epochs.

There are various optimizers available like minibatch gradient descent, gradient descent with momentum, RMSprop, and Adam. They all have the same performance on this dataset. Let us now observe the effect of change of learning rate using the minibatch gradient descent on the number of epochs required to train the model on full training set.

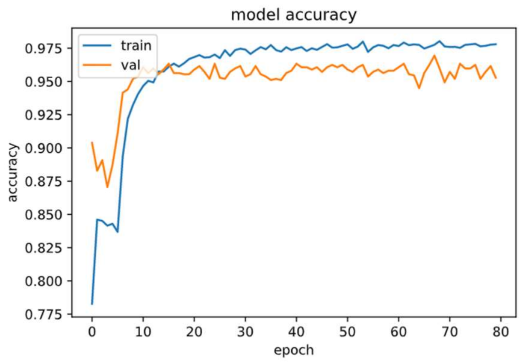
Learning Rate	No. of epochs
0.001	75
0.01	30
0.1	17

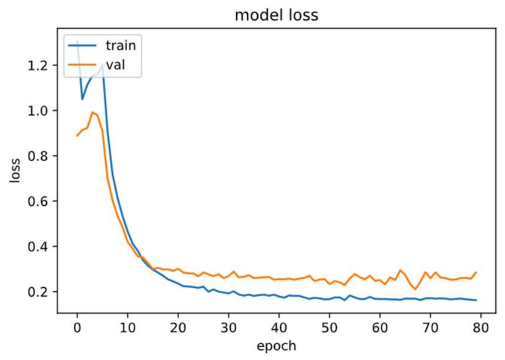
It can be observed that the number of epochs required to achieve appreciable accuracy is less for a higher learning rate. A higher learning rate speeds up the process of learning. However, in some situations, using a higher learning rate may not be helpful in reaching the minima. It's better to always lower the learning rate with epochs.

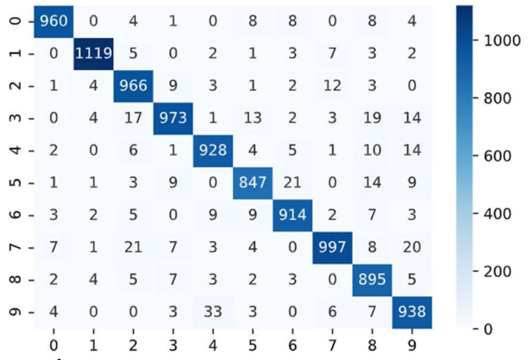
MNIST Dataset using other classifiers –

Neural Networks –

A neural network with two hidden layers consisting of 64 and 128 neurons each was designed. Adam optimization with initial learning rate 0.01 was used. The learning rate was reduced by a factor of 0.1 every time same model loss was obtained for 5 consecutive epochs. ReLU activation function was used. The training data was shuffled after every epoch. Dropout with probability 0.2 was applied in the hidden layers. Additionally, L2 regularization with parameter 0.001 was applied to remove overfitting. The model was trained for 80 epochs. Following are the model accuracy and loss curves.







	0	1	2	3	4	5	6	7	8	9
0	960	0	4	1	0	8	8	0	8	4
1	0	1119	5	0	2	1	3	7	3	2
2	1	4	966	9	3	1	2	12	3	0
3	0	4	17	973	1	13	2	3	19	14
4	2	0	6	1	928	4	5	1	10	14
5	1	1	3	9	0	847	21	0	14	9
6	3	2	5	0	9	9	914	2	7	3
7	7	1	21	7	3	4	0	997	8	20
8	2	4	5	7	3	2	3	0	895	5
9	4	0	0	3	33	3	0	6	7	938

The model achieved a training accuracy of 97.94%, a validation accuracy of 96.13% and a training accuracy of 95.37%. The test set confusion matrix has been shown on the right. Thus, it can be concluded that neural networks also give a very high accuracy like the CNNs for this dataset. But CNN still outperforms NN here by a slight margin. NN fail to catch temporal dependencies. Hence, CNNs are always preferrable for classifying image data.

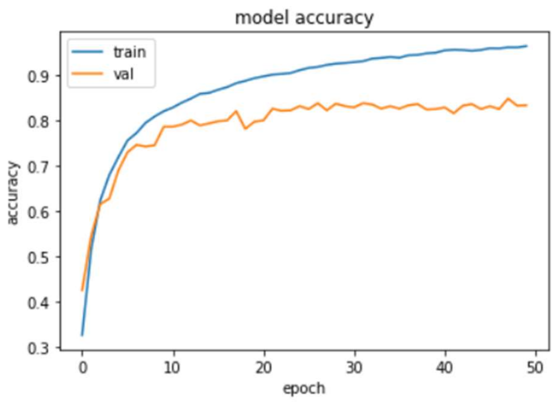
Support Vector Machines –

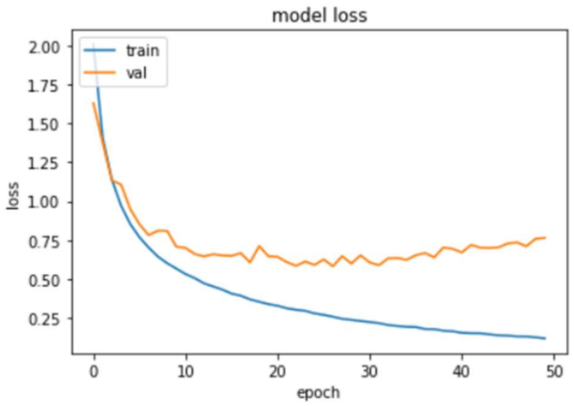
A SVM with Gaussian RBF kernel with parameter C = 1 was used. Grid search could not be used due to computational costs since the dataset is very huge and grid search would take large amount of time. The trained SVM model gives training accuracy of 98.54%. A test accuracy of 96.52% was obtained on the test set. The performance of this SVM was better than the performance of neural network which I have used. However, it fails to beat CNN. Like NN, SVMs also fail to capture temporal dependencies and hence, is not preferred for classifying image data.

MNIST-rot Dataset using CNN –

The training dataset consists of 12000 images. The test dataset consists of 10000 images. These images are rotated version of some of the samples of previous dataset (as shown on the right). All images are of shape 28x28. The data stored in '.mat' format. The training data was further divided to facilitate cross validation.

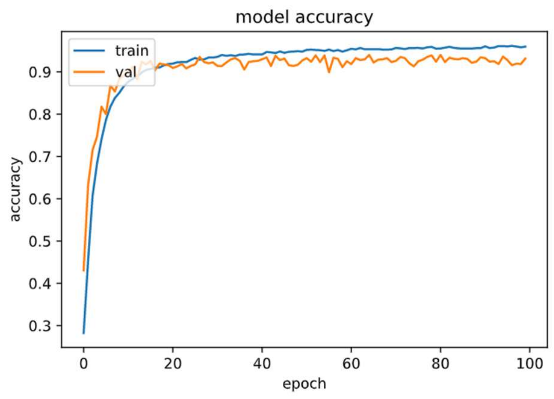
10% of the training data was used for validation. This is again a 10-class classification problem. Categorical cross entropy loss was used as objective function. Firstly, the same model as used in the previous part (with no regularization) has been trained for this case. Let us observe the following curves.

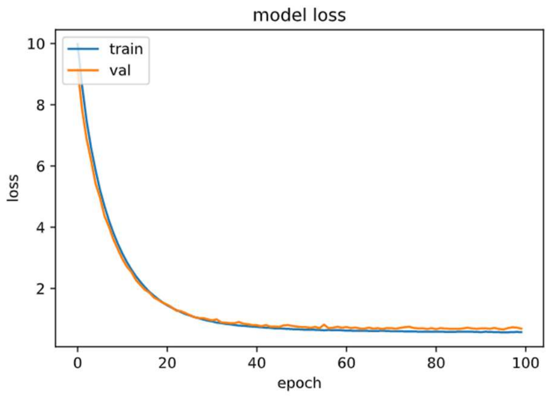


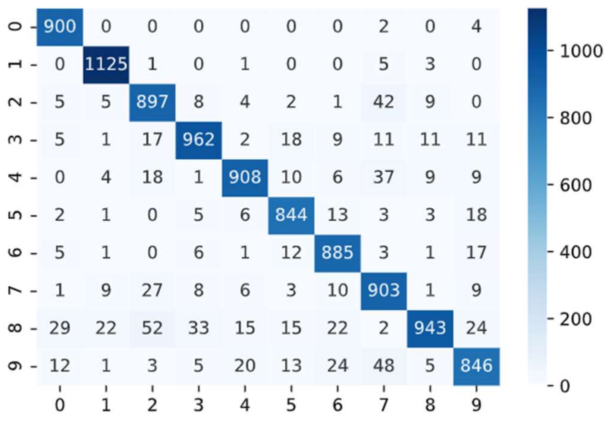


It is observed that the model overfits badly. The training accuracy shoots up to 97%, whereas validation accuracy is only 83%. The test accuracy is 81%. Let us apply now apply regularization to mitigate overfitting. After applying regularization, the model was underfitting, meaning it could learn well with training accuracy of around 70%. Thus, to learn the classifier, a deeper architecture has been explored, the details of which are shown in the table. This is because the model needs to learn complex features and it should correctly predict a digit even if it is rotated by certain degrees.

Layer	Hyper-parameters	Output Shape
Conv2D	Filters – 8, Shape – 3x3, Strides – 1, Padding – done	28x28x8
Conv2D	Filters – 16, Shape – 3x3, Strides – 1, Padding – done	28x28x16
MaxPool2D	Shape – 2x2, Strides – 1	14x14x16
Conv2D	Filters – 32, Shape – 3x3, Strides – 1, Padding – done	14x14x32
Conv2D	Filters – 64, Shape – 3x3, Strides – 1, Padding – done	14x14x64
MaxPool2D	Shape – 2x2, Strides – 1	7x7x64
Conv2D	Filters – 96, Shape – 3x3, Strides – 1, Padding – done	7x7x96
Conv2D	Filters – 128, Shape – 3x3, Strides – 1, Padding – done	7x7x128
MaxPool2D	Shape – 2x2, Strides – 1	3x3x128
Fully Connected	No. of neurons – 1152, Dropout – 0.2	1152
Fully Connected	No. of neurons – 64, Dropout – 0.2	64
Output		10







There are a total of 264,938 trainable parameters. L2 regularization with parameter 0.01 has been applied to prevent overfitting. In addition, a dropout with probability 0.2 was applied in both the fully connected layers. With Adam optimization algorithm (learning rate = 0.01), the model was trained for 100 epochs. The training accuracy obtained was 96.3% and validation accuracy of 93.15% was obtained. The test set accuracy was 92.13%. The test set confusion matrix is shown. Notice, the model loss curve. The validation loss follows the training loss almost always and hence the model does not overfit.

Let us now compare the performance of the model using various activation functions (ReLU and Leaky ReLU only), optimizers (SGD, RMSprop, and Adam), and different regularization techniques (dropout with the given probability applied only in the fully connected layers in all the models).

Optimizer (lr = 0.01)	Regularization	Activation Function	Training Accuracy	Validation Accuracy	Test Accuracy
SGD	L2 (0.01) + Dropout (0.2)	ReLU	93.28%	91.99%	90.13%
SGD	L2 (0.03) + Dropout (0.2)	Leaky ReLU (0.1)	82.2%	80.1%	78.33%
SGD	L1 (0.001) + Dropout (0.2)	ReLU	93.88%	90.55%	88.23%
SGD	L1(0.005) + Dropout (0.2)	Leaky ReLU (0.1)	71.88%	71.50%	68.12%
RMSprop	L2 (0.01) + Dropout (0.2)	ReLU	90.15%	90.64%	90.32%
RMSprop	L2 (0.03) + Dropout (0.2)	Leaky ReLU (0.1)	87.2%	83.93%	82.41%
RMSprop	L1 (0.001) + Dropout (0.2)	ReLU	91.74%	87.71%	88.84%
RMSprop	L1(0.001) + Dropout (0.6)	Leaky ReLU (0.1)	81.29%	77.97%	79.73%

Adam	L2 (0.01) + Dropout (0.2)	ReLU	96.3%	93.15%	92.13%
Adam	L2 (0.01) + Dropout (0.2)	Leaky ReLU (0.1)	87.27%	84.00%	81.08%
Adam	L1 (0.001) + Dropout (0.2)	ReLU	92.02%	90.74%	90.35%
Adam	L1(0.001) + Dropout (0.6)	Leaky ReLU (0.1)	95.53%	64.86%	55.87%

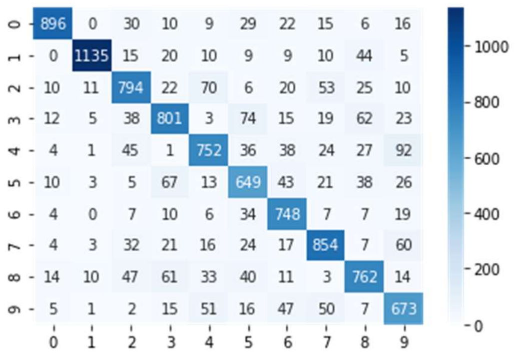
From the following table

- It can be observed that ReLU performs better than Leaky ReLU.
- Also, L2 regularization gives better results, prevents overfitting in a better way than L1. In case of L1 regularization, it was observed that dropout probability had to be increased to mitigate overfitting.
- When using ReLU, Adam gives better results than the others. Though there is not much noticeable difference, Adam optimizer helped in converging quicker and gave the best model (which has been discussed in detail before the table).

MNIST-rot Dataset using other classifiers –

Neural Networks –

Using the same neural network architecture along with same choices of optimizers, and other heuristics as used in the previous part (NN), the model was trained on this dataset which came out to be overfit. Hence, after applying more regularization, the same model after training for 100 epochs gave a training accuracy of 87.91%, but the validation and test accuracies were around 82%. The model accuracy and loss curve, and confusion matrix has been shown.



	0	1	2	3	4	5	6	7	8	9
0	896	0	30	10	9	29	22	15	6	16
1	0	1135	15	20	10	9	9	10	44	5
2	10	11	794	22	70	6	20	53	25	10
3	12	5	38	801	3	74	15	19	62	23
4	4	1	45	1	752	36	38	24	27	92
5	10	3	5	67	13	649	43	21	38	26
6	4	0	7	10	6	34	748	7	7	19
7	4	3	32	21	16	24	17	854	7	60
8	14	10	47	61	33	40	11	3	762	14
9	5	1	2	15	51	16	47	50	7	673

Question 4

The dataset consists of fMRI recording of 34 patients with Alzheimer's disease, and 47 normal subjects in rest state. The problem is two-class classification. There are two parcellations for each sample – 'aal' and 'power' of dimension (140, 116) and (140, 264) respectively. Thus, there are a total of 81 samples in the dataset.

Classification using Support Vector Machines –

64 samples have been used for training and 17 for testing purposes. Cross validation grid search over C and gamma have been done to get the best fit model. All the kernels have been tried.

Parcellation – 'power'

The feature vector has been reshaped into a 36960-dimensional feature vector.

Kernel	Training accuracy	Test accuracy
Gaussian RBF	59.4%	52.94%
Polynomial	59.4%	52.94%
Linear	56.63%	52.94%
Sigmoidal	56.25%	52.94%

Parcellation – 'aal'

The feature vector has been reshaped into a 16240-dimensional feature vector

Kernel	Training accuracy	Test accuracy
Gaussian RBF	59.4%	52.94%
Polynomial	59.4%	52.94%
Linear	60.89%	58.8%
Sigmoidal	53.1%	47.05%

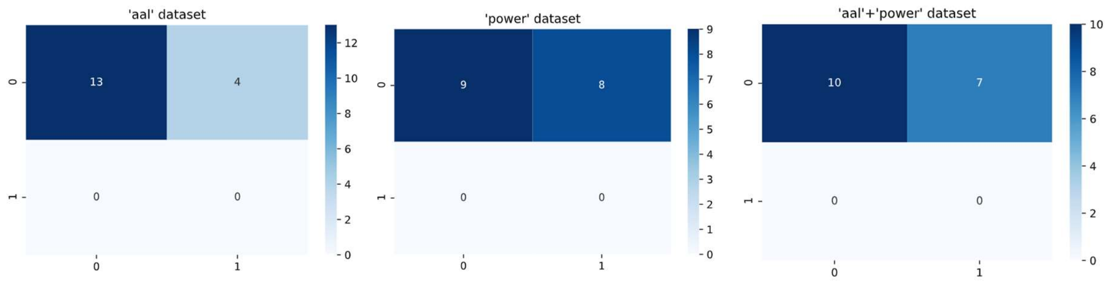
It can be observed that the SVM kernels have performed poorly. The accuracies obtained are not much better than randomly guessing the label. The number of samples are too less, and the number of features is very large. I tried to apply SVM after reducing dimension using PCA, but it also gave poor results. Also, after examining the confusion matrices for each kernel, it was clear that the classifier was not learning any features. In fact, it classified all the test samples into one of the classes.

Classification using CNNs –

Due to scarcity of data, no validation set was considered. 64 samples have been used for training and 17 for testing purposes. Various architectures of CNNs were tried. I altered the loss functions, added regularization (both L2 and dropout), changed activation functions, applied heuristics such as batch normalization. However, none of the models gave appreciable accuracy. The training accuracy reached 100% in few epochs even after applying a lot of regularization. Since we are dealing with medical data which are not even images, data augmentation should not be done. However, I applied data augmentation techniques such as rotation, and flipping to increase the size of the dataset. This was done using an inbuilt Tensorflow function. The results were still not satisfactory. A 5-layer (Convolution + Maxpool) CNN model was designed which gave the accuracies mentioned in the table for each dataset.

Dataset	Training accuracy	Test accuracy
‘aal’	53.19%	76.47%
‘power’	59.68%	52.94%
‘aal’ + ‘power’	65.03%	58.88%

The accuracy metric can be fictitious. Let us check the confusion matrix. From the below matrices, it can be concluded that all the test samples were classified into one class which meant that the models did not learn any feature at all.



Classification using feedforward neural networks –

The data was flattened to convert into 1D data which can be fed into a feedforward neural network. Within a few epochs, the training accuracy reached 100% whereas the test accuracy hovered around 50% . After applying a high value of L2 and dropout regularization, the training accuracy obtained was closer to the test accuracy but then it was observed the confusion matrices obtained were like how they were in case of CNN. Hence, the model did not learn at all and instead classified all the test samples in one class.

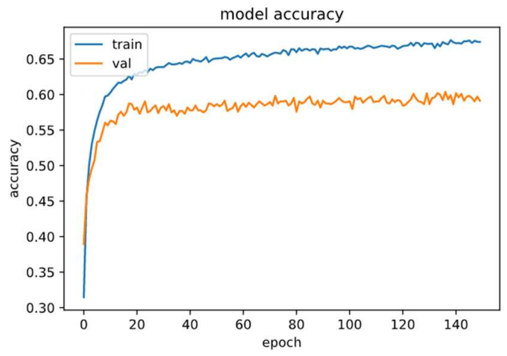
Another reason why these models do not perform well is that the data we have is dependent on time. It is a recording of activities in different brain region as a function of time. There are temporal dependencies which are captured very well by recurrent networks and not by other models. Recurrent neural network models such as LSTMs may work better in this case. However, insufficient data is still the major reason why the models cannot learn the features or patterns in the data since an LSTM trained on this dataset gave same kind of results as given by CNN or feedforward neural networks.

Question 5

The data consists of 11500 samples with 178 features each. The problem is 5-class classification. As a part of data preprocessing, the features were normalized, and the output labels were encoded into a 'one hot vector' representation.

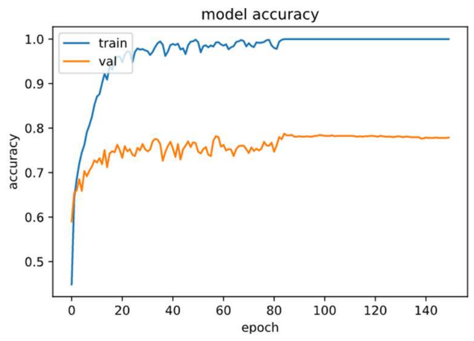
Classification using feedforward neural networks –

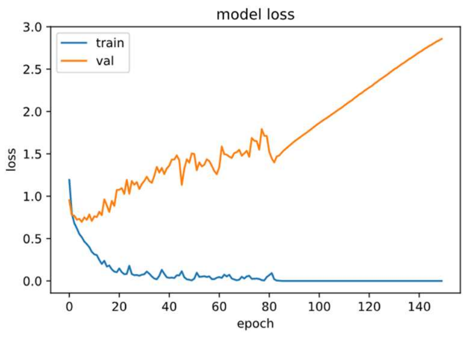
The dataset was split into three parts, 80% for training, 10% for cross validation, and 10% for testing. Categorical cross entropy loss was used as objective function for minimization. Adam optimization has been used. Batch size of 32 was used. Let us explore Neural networks first.



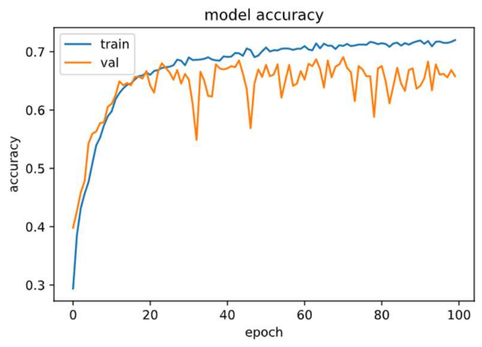
A simple neural network with one hidden layer containing 16 neurons was first trained. From the Accuracy vs epoch graph above, it is observed that training accuracy is 65% whereas the validation accuracy is less than 60%. The test accuracy comes out to be 62%. There is a need of more complex model to learn complex patterns from data.

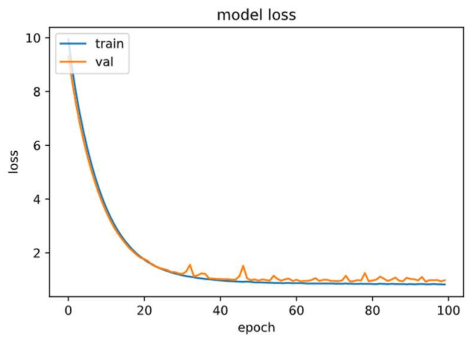
The deeper NN model now has three hidden layers with 128, 256, and 32 neurons. From the below two graphs, it can be concluded that the NN model is overfitting. The train accuracy reaches 100% but validation and test accuracy does not cross 80%. The validation loss does not follow the training loss. Hence, regularization must be performed to reduce overfitting.

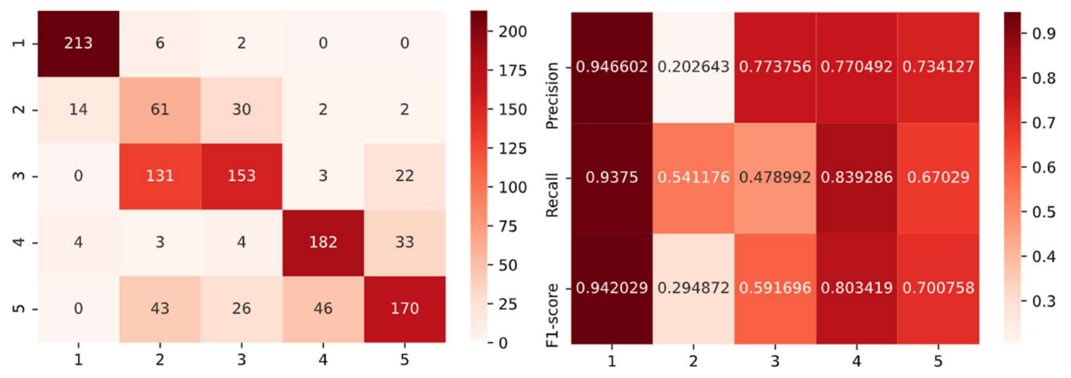




L2-regularization with parameter 0.01 and dropout with probability of dropping neurons 0.1 has been applied in each layer in the previous NN. Following are the model accuracy, and model loss curves, and the confusion matrix obtained on the test set.







From the model accuracy and loss curves, it can be concluded that the model does not overfit. Now, let us see the confusion matrix and the precision-recall-F1-score matrix. Class 1 samples seem to get classified very accurately as confirmed by high F1-score. However, the model seems to confuse between classes 2 and 3 and hence a very low F1-score is obtained for both. Class 4 has relatively good precision and recall. Class 5 also gets misclassified often.

Let us see the effect of increasing depth of NN. The following is a table which shows the effect of depth on the performance. Regularization have been applied whenever overfitting happened. Notice how the accuracy increase with increase in network size and increased parameters.

Number of hidden layers in the NN	Number of neurons	Number of trainable parameters	Regularization used	Training Accuracy	Validation Accuracy	Test Accuracy
1	16	2949	No	65.35%	62.42%	57.3%
2	64, 128	20,421	L2 – 0.01	71.3%	66.28%	65.56%
3	128, 256, 32	65,381	L2 – 0.01, Dropout – 0.1	73.42%	68.89%	67.74%

Feedforward neural networks could not produce good results since the data is in the form of a time series. Hence, advanced classifiers suited for time series data such as recurrent neural network models, LSTMs, or GRUs can be used to better classify the samples in this dataset.

Classification using Support Vector Machines –

Now, the dataset is split into 80%-20% for training and testing purposes. Hyper-parameter grid search like in the previous problems have been done here.

Kernel	Training Accuracy	Test Accuracy
Gaussian RBF	76.9%	66.52%
Polynomial	39.1%	40.87%
Linear	33.53%	26.61%
Sigmoidal	28.53%	30.26%

Let us explore Gaussian RBF kernel for various choices of hyper-parameters C and gamma.

Hyper-parameter	Training accuracy	Test accuracy
C = 1, gamma = 0.1	91.02%	65.43%
C = 1, gamma = 0.01	64.05%	60.04%
C = 1, gamma = 0.001	46.20%	45.48%
C = 1, gamma = 0.0001	34.27%	32.65%
C = 10, gamma = 0.1	99.56%	66.13%
C = 10, gamma = 0.01	76.9%	66.52%
C = 10, gamma = 0.001	54.73%	52.65%
C = 10, gamma = 0.0001	41.38%	39.69%
C = 100, gamma = 0.1	100%	65.39%
C = 100, gamma = 0.01	91.45%	70.39%
C = 100, gamma = 0.001	65.33%	60.48%
C = 100, gamma = 0.0001	47.37%	44.83%
C = 1000, gamma = 0.1	100%	65.39%
C = 1000, gamma = 0.01	98.44%	68.39%
C = 1000, gamma = 0.001	77.30%	66.52%
C = 1000, gamma = 0.0001	55.26%	51.56%

Observations –

- Gaussian kernel performed way better than the other three kernels. Polynomial kernels perform second best. The other two classifiers perform poor.
- Considering Gaussian kernel, the SVM classifier overfits when the gamma value is large. As we decrease the gamma value, the model starts to perform well. However, after a certain threshold, the classifier starts to underfit. Thus, the gamma parameter works like a regularization parameter.
- The Gaussian RBF kernel SVM performs a little better than the neural networks in terms of training accuracy. However, a higher difference between the training and test accuracy in SVM case indicate that the model is a little bit more overfit that neural network models.