Pattern Recognition and Neural Networks Assignment - 2 Report

Apoorv Pandey, Kaustuv Ray, Samrat Yadav, Pratyush Gauri
IISc Bangalore

In this Assignment, we will implement different recent ML Algorithms on Binary Classification problem, Multi-Class Classification Problem, Bounding box regression Problem, Frame classification on audio data, Generative Models respectively as described in our explanation from Q1 to Q5.

I. Support Vector Machines

We have implemented the SVM with and without Slack variables on each of the classification problems defined in Assignment - 1 and dataset used here is pneumoniamnist, bloodmnist and TIMIT dataset for binary, multiclass and audio classification respectively.

A. Hyper-parameters Tuning

In SVM formulation the hyperpararameter C (regularizer term) is varied from 0 to 5. C=0 represents that SVM is being operated without slack variable whereas C greater than that shows the formulation with slack variable.
We have also varied the kernel function of the SVM to see how the results vary with different kernels. In this problem, we have used the linear, polynomial, sigmoid and radial basis function (rbf) Kernel to operate on the dataset.
ALso the evaluation metric Accuracy, F1 Score and AUC score is used for analysation of the model performance and is tabulated in this report.

B. Observations

Pneumoniamnist Dataset:- It can be clearly observed that as we tend to provide some slack variable for SVM formulation, we do achieve a better accuracy for each of the kernels associated. But as the slack variable value is increased more, the accuracy reduces as we will not get better optimization. So, we do multiple experimentation to achieve the optimal parameters.
The Evaluation metric (kernel = 'linear', C = 1) is formed and out of 624 samples, the true positive samples are observed to be 127, false positive as 107, true negatives as 382 and false negatives of only 8. This shows that the model is capable of detecting the disease but not very precisely. But the model will not fail to identify the disease as the false negative rate is very low.

Bloodmnist Dataset:-
Similar observations comparing the model with and without slack variables. The Evaluation metric (kernel = 'rbf', C = 1) is formed and out of 624 samples, the true positive samples are observed to be 147, false positive as 87, true negatives as 388 and false negatives of only 2. This shows that the model is capable of detecting the disease but not very precisely. But the model will not fail to identify the disease as the false negative rate is very low.

TIMIT Dataset:-
Similar observations comparing the model with and without slack variables. The Evaluation metric (kernel = 'rbf', C = 1) is formed and out of 1000 samples, the true positive samples are observed to be 240, false positive as 110, true negatives as 594 and false negatives of only 56. This shows that the model is capable of detecting the vowels but not very precisely. But the model will not fail to identify the vowels as the false negative rate is very low.

II. GridSearch Method

Grid search is essentially an optimization algorithm which lets you select the best parameters for your optimization problem from a list of parameter options that you provide, hence automating the 'trial-and-error' method. Although it can be applied to many optimization problems, but it is most popularly known for its use in machine learning to obtain the parameters at which the model gives the best accuracy.
We have applied this method on each of the kernels existing and given the grid search different regularising parameters C: [1, 10, 100, 1000] and for the 'rbf' kernel, we have given the gamma value as 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9].
It is observed that for pneumoniamnist data-set the optimal kernel is 'polynomial' and the best regulariser value is C = 1000. The bloodmnist dataset also has best kernel as 'polynomial' and best value of regulariser as C = 1000. For TIMIT dataset, we see unique result of 'rbf' as the optimal kernel and C = 100 and gamma = 0.8 as the best parameters.

III. Compare Fisher LD and SVMs

In this problem we will compare the results of Fisher LD and SVMs with different kernels(Linear, polynomial, sigmoid, rbf). We will compare the results across the following datasets- PneumoniaMNIST, BloodMNIST, TIMIT dataset.

A. Comparing over PneumoniaMNIST

- From Table. IV its clear that - $Sigmoid < Linear < Polynomial < FLDA < Rbf$

TABLE I: SVM Implementation with and without Slack variables on BloodMNIST

Kernel	Linear			Polynomial			Sigmoid			Rbf		
	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0
Accuracy	0.854	0.815	0.811	0.655	0.796	0.815	0.79	0.735	0.734	0.625	0.857	0.862
F1	0.894	0.869	0.865	0.784	0.859	0.87	0.85	0.808	0.806	0.77	0.897	0.9
AUC	0.810	0.761	0.758	0.54	0.73	0.757	0.735	0.683	0.683	0.5	0.811	0.819

TABLE II: SVM Implementation with and without Slack variables on PneumoniaMNIST

Kernel	Linear			Polynomial			Sigmoid			Rbf		
	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0
Accuracy	0.796	0.815	0.811	0.67	0.817	0.835	0.79	0.735	0.734	0.7	0.857	0.862
F1	0.858	0.869	0.865	0.79	0.872	0.882	0.85	0.808	0.806	0.8	0.897	0.9
AUC	0.733	0.761	0.758	0.56	0.758	0.783	0.735	0.683	0.683	0.6	0.811	0.819

TABLE III: SVM Implementation with and without Slack variables on TIMIT dataset

Kernel	Linear			Polynomial			Sigmoid			Rbf		
	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0	C = 0.0	C = 1.0	C = 5.0
Accuracy	0.813	0.813	0.79	0.645	0.781	0.809	0.65	0.533	0.51	0.65	0.834	0.841
F1	0.864	0.862	0.845	0.782	0.85	0.867	0.79	0.641	0.63	0.79	0.877	0.882
AUC	0.77	0.774	0.75	0.5	0.705	0.744	0.5	0.486	0.45	0.5	0.8	0.811

TABLE IV: Comparison over PneumoniaMNIST

Kernel	Linear(C=1.0)	Polynomial(C=5.0)	Sigmoid(C=0.0)	Rbf(C=5.0)	Fisher LDA
Accuracy	0.815	0.835	0.79	0.862	0.836
F1	0.869	0.882	0.85	0.9	0.882
AUC	0.761	0.783	0.735	0.819	0.789

TABLE V: Comparison over BloodMNIST

Kernel	Linear(C=1.0)	Polynomial(C=5.0)	Sigmoid(C=0.0)	Rbf(C=5.0)	Fisher LDA
Accuracy	0.854	0.835	0.79	0.862	0.656
F1	0.894	0.882	0.85	0.9	0.643
AUC	0.810	0.783	0.735	0 .819	0.671

TABLE VI: Comparison over TIMITdataset

Kernel	Linear	Polynomial	Sigmoid	Rbf	Fisher LDA
True positive	0.226	0.158	0.116	0.24	0.548
True negative	0.587	0.623	0.417	0.594	0.270
False positive	0.124	0.192	0.234	0.11	0.082
False negative	0.063	0.027	0.233	0.056	0.099

TABLE VII

Algorithm	L1 Regression [MSE,mIoU, MAE]	L2 Regression	Elastic Net(L1 ratio = 0.9)
Reg-Rate = 0.001	[20.378,0.0,4.746]	[20.382,0.0,4.746]	[20.378,0.0,4.746]
Reg-Rate = 0.01	[20.374,0.0,4.745]	[20.380,0.0,4.746]	[20.375,0.0,4.745]
Reg-Rate = 0.1	[20.368,0.0,4.744]	[20.378,0.0,4.745]	[20.369,0.0,4.744]

- Fisher LDA performs better than SVMs with linear, polynomial and sigmoid kernels.
- SVM with Rbf kernel performs the best of all the models.

B. Comparing over BloodMNIST

- From Table. V its clear that - $FLDA < Sigmoid < Polynomial < Linear < Rbf$
- All the SVMs perform better than Fisher LDA.
- SVM with Rbf kernel performs the best.

C. Comparing over TIMIT dataset

- From Table. VI its clear that - TP: $Sigmoid < Polynomial < Linear < Rbf < FLDA$
  TN: $FLDA < Sigmoid < Linear < Rbf < Polynomial$
  FP: $FLDA < Rbf < Linear < Polynomial <$

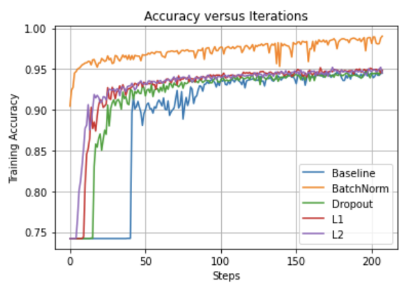

Fig. 1: Training Acc vs iter plot for single layer on PneumoniaMNIST

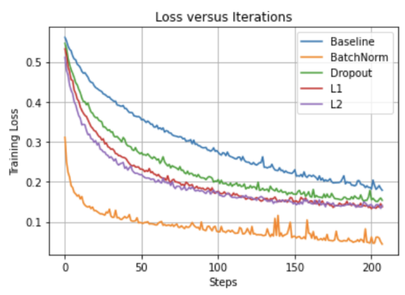

Fig. 2: Training Loss vs iter plot for single layer on PneumoniaMNIST

$Sigmoid$
FN: $Polynomial < Rbf < Linear < FLDA < Sigmoid$

- In case of TN and FN, Fisher LDA performs the worst.
- In case of TP and FP, Fisher LDA performs quite well.

D. Conclusion

SVM with Rbf kernel always performs better than Fisher LD and SVM with other kernels. Fisher LD performed well in PneumoniaMNIST and TIMIT dataset as compared to BloodMNIST. In BloodMNIST Fisher LD performed the worst.

IV. Multi Layer Perceptron

A. Pneumoniamnist Dataset

1) Single Layer Architecture: We applied the following architectures on this question:-
Baseline - [Linear ,Sigmoid ,Linear,Sigmoid]
BatchNorm - [Linear, BatchNorm, Sigmoid,Linear,Sigmoid]
Dropout - [Linear, Sigmoid,Dropout,Linear,Sigmoid]
L1 - [Linear,Sigmoid,Linear,Sigmoid]
L2 - [Linear, Sigmoid, Linear, Sigmoid]

We used 100 neurons in the hidden layer. In the dropout layer, we dropped 20 percent of the activations. In the L1 and L2 architectures, we used a linear layer with 0.01 as the regularization rate. I used a batch size of 100. We used Adam optimizer to perform gradient descent.

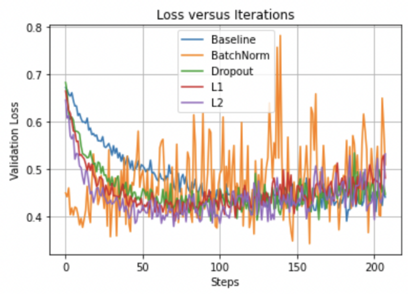

Fig. 3: Validation Loss vs iter plot for single layer on PneumoniaMNIST

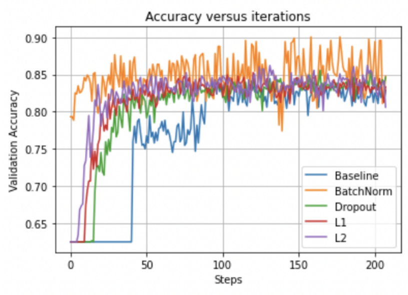

Fig. 4: Validation Acc vs iter plot for single layer on PneumoniaMNIST

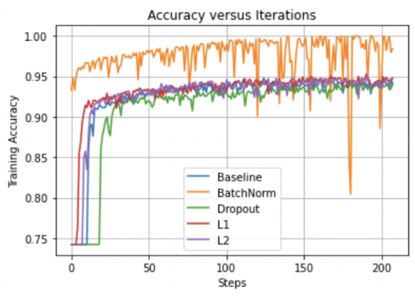

Fig. 5: Training Acc vs iter plot for two layer on PneumoniaMNIST

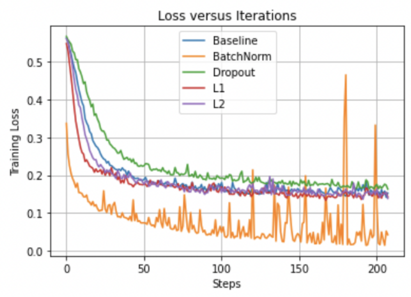

Fig. 6: Training Loss vs iter plot for two layer on PneumoniaMNIST

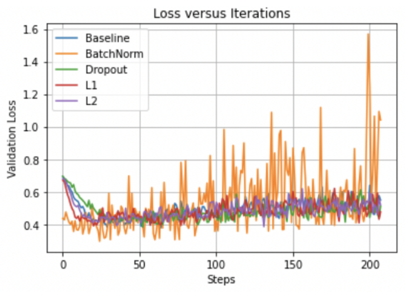

Fig. 7: Validation Loss vs iter plot for two layer on PneumoniaMNIST

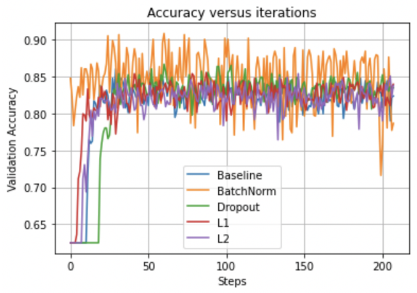

Fig. 8: Validation Acc vs iter plot for two layer on PneumoniaMNIST

2) Two Layer Architecture: We applied the following architectures on this question:-
Baseline - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]
BatchNorm - [Linear,BatchNorm,Sigmoid,Linear,BatchNorm,Sigmoid,Linear,
Sigmoid]
Dropout - [Linear,Sigmoid,Dropout,Linear,Sigmoid,Dropout,
Linear,Sigmoid]
L1 - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]
L2 - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]

We used 100 neurons in each hidden layer. In the dropout layer we dropped 20 percent of the activations. In the L1 and L2 architectures, we used a batch size of 100. We used a linear layer with 0.01 as the regularization rate.The represented table gives the metrics of these architectures.

3) Observations and Conclusions: Adding an extra layer has improved the accuracy score of the models indicating the nonlinear nature of the dataset. Batch Norm significantly increases training accuracy over other architectures . Also the batch norm validation plots are very noisy and this may depend on the batch size chosen. Smaller batch sizes are supposed to be more noisy and bigger batch sizes will be more similar to the whole dataset.

B. Bloodmnist Dataset

1) Single Layer Architecture: We applied the following architectures on this question:-
Baseline - [Linear ,Sigmoid ,Linear,Sigmoid]
BatchNorm - [Linear, BatchNorm, Sigmoid,Linear,Sigmoid]
Dropout - [Linear, Sigmoid,Dropout,Linear,Sigmoid]
L1 - [Linear,Sigmoid,Linear,Sigmoid]
L2 - [Linear, Sigmoid, Linear, Sigmoid]

We used 100 neurons in the hidden layer. In the dropout layer, we dropped 20 percent of the activations. In the L1 and L2 architectures, We used a linear layer with 0.01 as the regularization rate. We used a batch size of 100.The following table gives the metrics of these architectures.

2) Two Layer Architecture: We tried the following 2 hidden layer architectures:-

Baseline - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]
BatchNorm - [Linear,BatchNorm,Sigmoid,Linear,BatchNorm,
Sigmoid,Linear,Sigmoid]
Dropout - [Linear,Sigmoid,Dropout,Linear,Sigmoid,Dropout,
Linear,Sigmoid]
L1 - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]
L2 - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]

We used 100 neurons in each hidden layer. In the dropout layer we dropped 20 percent of the activations. In the L1 and L2 architectures, we used a linear layer with 0.01 as the regularization rate. We used a batch size of 100. The following table gives the metrics of these architectures.

3) Observations and Conclusion: All the models except batch norm are performing very poorly. Batch Norm is giving decent results and surpasses the accuracy obtained in the previous assignment using Linear Models. Also the accuracy of all architectures increases on adding an additional layer which indicates the highly non-linear nature of the dataset.

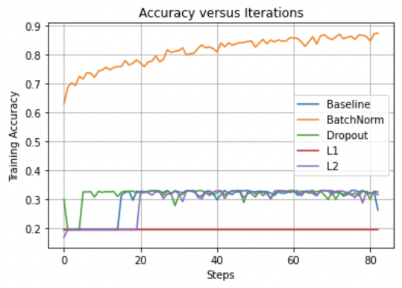

Fig. 9: Training Acc vs iter plot for single layer on BloodMNIST

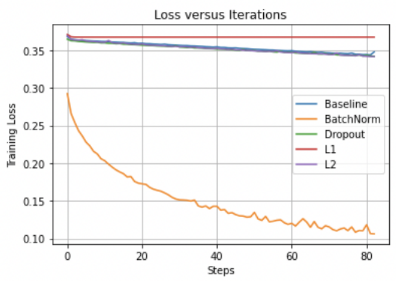

Fig. 10: Training Loss vs iter plot for single layer on BlooodMNIST

TABLE VIII: Evaluation Metrics for Single hidden layer over Pneumoniumnist Dataset

Models	Baseline	BatchNorm	Dropout	L1	L2
Accuracy	0.822	0.847	0.846	0.833	0.806
F1	0.872	0.890	0.888	0.878	0.864
AUC	0.770	0.801	0.802	0.789	0.744

TABLE IX: Evaluation Metrics for Two hidden layer over Pneumoniumnist Dataset

Models	Baseline	BatchNorm	Dropout	L1	L2
Accuracy	0.830	0.798	0.834	0.839	0.838
F1	0.878	0.859	0.880	0.882	0.881
AUC	0.778	0.734	0.789	0.797	0.797

TABLE X: Evaluation Metrics for Single hidden layer over Bloodmnist Dataset

Models	Baseline	BatchNorm	Dropout	L1	L2
Accuracy	0.258	0.813	0.323	0.194	0.312
F1	0.105	0.792	0.136	0.040	0.131
AUC	0.137	0.796	0.171	0.074	0.227

TABLE XI: Evaluation Metrics for Single hidden layer over TIMIT Dataset

Models	Baseline	BatchNorm	Dropout	L1	L2
True Positive	0.579	0.582	0.583	0.581	0.578
False Positive	0.072	0.097	0.073	0.070	0.069
True Negative	0.281	0.255	0.279	0.282	0.282
False Negative	0.067	0.065	0.064	0.065	0.068

TABLE XII: Evaluation Metrics for Two hidden layer over TIMIT Dataset

Models	Baseline	BatchNorm	Dropout	L1	L2
True Positive	0.584	0.603	0.579	0.582	0.577
False Positive	0.068	0.111	0.070	0.064	0.068
True Negative	0.284	0.241	0.282	0.288	0.284
False Negative	0.063	0.044	0.068	0.065	0.070

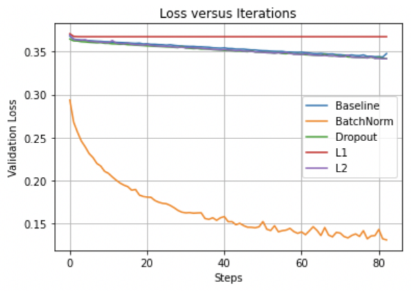

Fig. 11: Validation Loss vs iter plot for single layer on BloodMNIST

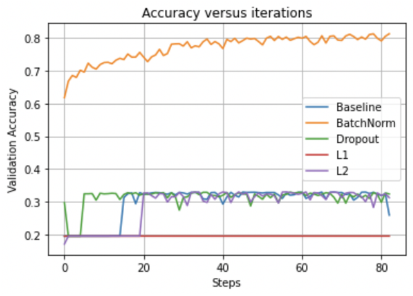

Fig. 12: Validation Acc vs iter plot for single layer on BloodMNIST

C. Bounding Box Regression Problem

Data Preprocessing: We resized the image to a target size = (447,300). I resized the bounding boxes accordingly.
We performed the following Data Augmentation techniques - Random Crop, Random Rotation of the image.

Architecture: We added a Sequential layer(nn.Sequential(nn.BatchNorm1d(512), nn.Linear(512, 4))) after the last layer in both resnet 34 and VGG 19 models.

D. TIMIT Dataset

1) Data preprocessing: We transformed the audio and used MFCC's as the feature input x. Also in order to ensure all inputs have the same length I used zero padding on MFCC.

2) Single Hidden Layer Architectures: We tried the following architecture:-
Baseline - [Linear,Sigmoid,Linear,Sigmoid]

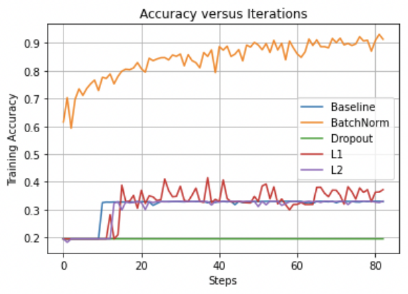

Fig. 13: Training Acc vs iter plot for two layer on BloodMNIST

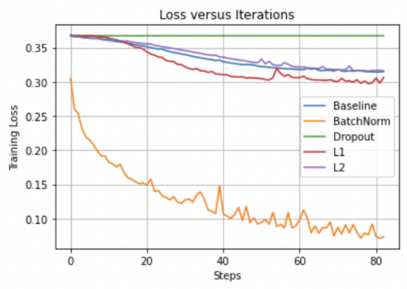

Fig. 14: Training Loss vs iter plot for two layer on BloodMNIST

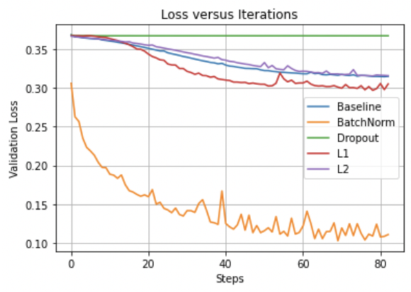

Fig. 15: Validation Loss vs iter plot for two layer on BloodMNIST

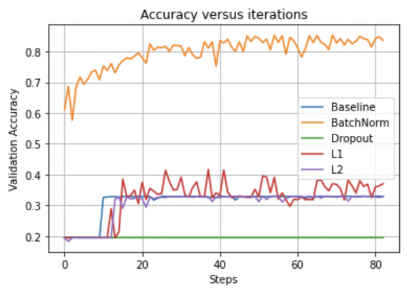

Fig. 16: Validation Acc vs iter plot for two layer on BloodMNIST

BatchNorm - [Linear,BatchNorm,Sigmoid,Linear,Sigmoid]
Dropout - [Linear,Sigmoid,Dropout,Linear,Sigmoid]
L1 - [Linear,Sigmoid,Linear,Sigmoid]
L2 - [Linear,Sigmoid,Linear,Sigmoid]

We used 100 neurons in the hidden layer. In the dropout layer, I dropped 20 percent of the activations. In the L1 and L2 architectures, we used a linear layer with 0.01 as the regularization rate. We used 100 as the batch size. The represented table gives the metrics of these architectures.

3) *Two Hidden Layer Architectures:* We tried the following architecture:-
Baseline - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]
BatchNorm - [Linear,BatchNorm,Sigmoid,Linear, BatchNorm,Sigmoid,Linear,Sigmoid]
Dropout - [Linear,Sigmoid,Dropout,Linear,Sigmoid, Dropout,Linear,Sigmoid]
L1 - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]
L2 - [Linear,Sigmoid,Linear,Sigmoid,Linear,Sigmoid]

We used 100 neurons in each hidden layer. In the dropout layer I dropped 20 percent of the activations. In the L1 and L2 architectures, we used a linear layer with 0.01 as the regularization rate.I used a batch size of 100.The table gives the metrics of these architectures.

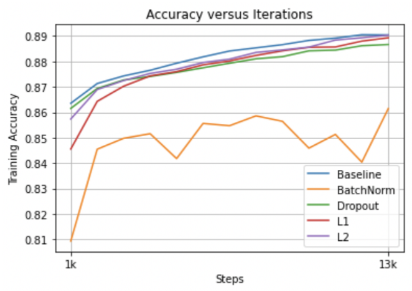

Fig. 17: Training Acc vs iter plot for single layer on TIMIT Dataset

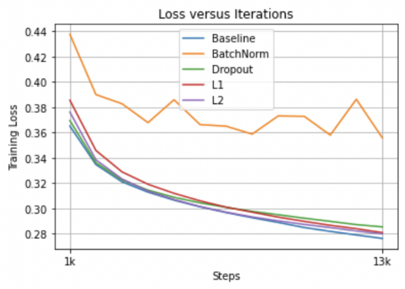

Fig. 18: Training Loss vs iter plot for single layer on TIMIT Dataset

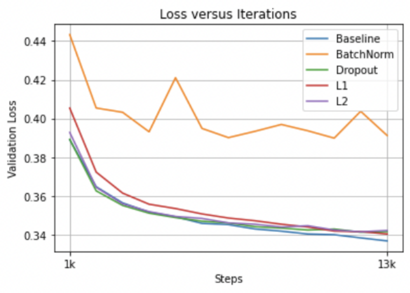

Fig. 19: Validation Loss vs iter plot for single layer on TIMIT Datasetv

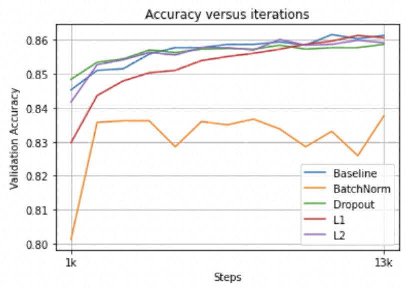

Fig. 20: Validation Acc vs iter plot for single layer on TIMIT Dataset

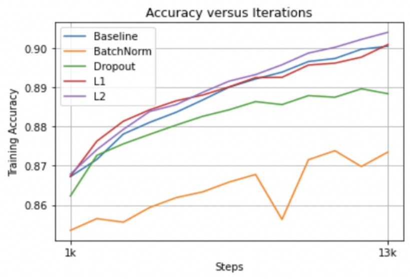

Fig. 21: Training Acc vs iter plot for two layer on TIMIT Dataset

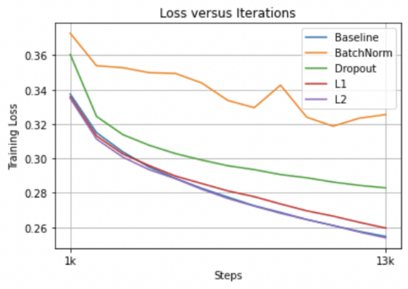

Fig. 22: Training Loss vs iter plot for two layer on TIMIT Dataset

4) Observations and Conclusions: From the above plots we can see that batch Norm is performing worse than other models. This may be due to the batch size chosen. Since the efficiency of batch norm depends on how good representative the chosen batch is of the whole training data set. Increasing the batch size in the batch norm might lead to better results. Also, dropout is performing worse than non-regularized models implying that test distribution is roughly the same as training distribution and there is not much need of regularization.

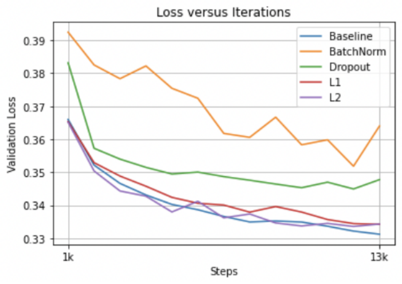

Fig. 23: Validation Loss vs iter plot for two layer on TIMIT Dataset

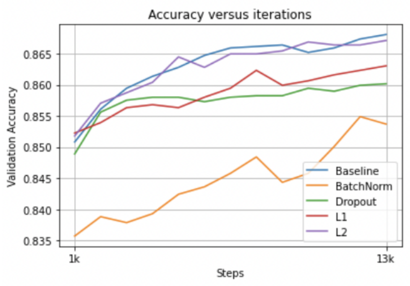

Fig. 24: Validation Acc vs iter plot for two layer on TIMIT Dataset

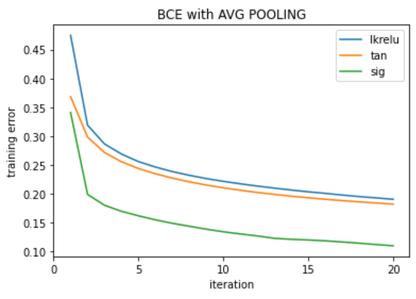

Fig. 25: CNN PneumoniaMNIST

V. CNN USING NUMPY

In this problem we will construct a simple CNN of 3 layers using numpy and code up backpropagation on it.

A. *Binary Classification problem(PneumoniaMNIST)*

Data preprocessing: The class labels are converted to one hot encoding

CNN architecture:

Convolution layer, Activation, Pooling layer,

Convolution layer, Activation, Pooling layer,

Reshape layer, Dense layer, Sigmoid

Experiments: For our experiments we will vary the Activation over Tanh, Leaky ReLU and Sigmoid functions, and vary the Pooling layer over Max pool and Average pool. For this binary classification problem we will use two loss functions - Binary cross entropy loss and Mean square error.

The kernel size is 3x3 and the stride is 1 The models were trained for 20 iterations. The plots (Fig. 25,26,27,28) show the training error for different CNN against the number of iterations. We varied the learning rates over [0.01, 0.03, 0.001, 0.003]. We ran multiple models with different learning rates, and selected the models which had the highest accuracy on the validation set. The validation accuracy, test accuracy(Table.XIII), AUC(Table. XV), F1 scores(Table.XIV) for different models are shown below.

Observations: For the PneumoniaMNIST data we observe that a 3 layer CNN with sigmoid activation function, binary

TABLE XIII: CNN Accuracy(PneumoniaMNIST)

Loss function and pooling	sigmoid	tanh	Leaky ReLU
MSE, Max pool	0.820512	0.815705	0.625
MSE, Avg pool	0.858974	0.852564	0.798076
BCE, Max pool	0.862179	0.625	0.849358
BCE, Avg pool	0.868589	0.875	0.713141

TABLE XIV: CNN F1 Score(PneumoniaMNIST)

Loss function and pooling	sigmoid	tanh	Leaky ReLU
MSE, Max pool	0.872727	0.869762	0.769230
MSE, Avg pool	0.896226	0.892271	0.858744
BCE, Max pool	0.897129	0.769230	0.889671
BCE, Avg pool	0.903529	0.904645	0.808556

TABLE XV: CNN AUC(PneumoniaMNIST)

Loss function and pooling	sigmoid	tanh	Leaky ReLU
MSE, Max pool	0.765811	0.759401	0.5
MSE, Avg pool	0.820512	0.811111	0.736752
BCE, Max pool	0.829059	0.5	0.808547
BCE, Avg pool	00.829914	0.850427	0.627777

TABLE XVI: CNN Metrics(BloodMNIST)

Metrics	Tanh Max pool	Sigmoid Max pool	Lk ReLU Max pool	Tanh Avg pool	Sigmoid Avg pool	Lk ReLU Avg pool
accuracy	0.746977	0.795078	0.765217	0.756383	0.795078	0.781715
F1 score	0.080937	0.009421	0.108524	0.033190	0.009421	0.016267
AUC	0.487196	0.397539	0.472247	0.378191	0.397539	0.439519

TABLE XVII: CNN Metrics(Traffic dataset)

Metrics	Tanh Max pool	Sigmoid Max pool	Lk ReLU Max pool	Tanh Avg pool	Sigmoid Avg pool	Lk ReLU Avg pool
MSE	1147.31998	1126.688	835.28321	1241.7159	1011.6476	1172.9883
mIoU	0.0004288	0.000428	0.0237023	0.0003864	0.000439	0.0002865
MAE	44.400224	43.99846	35.898297	46.270847	40.223261	44.608384

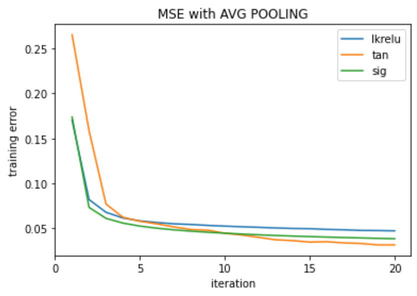

Fig. 26: CNN PneumoniaMNIST

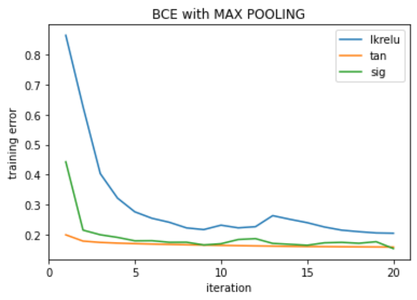

Fig. 27: CNN PneumoniaMNIST

cross entropy loss and average pooling gives the best accuracy. All other models perform quite decently. For F1 scores, CNN with tanh activation function, binary cross entropy loss and average pooling performed the best. For AUC, CNN with tanh activation function, binary cross entropy loss and average pooling performed the best.

B. Multi-Class Classification Problem (BloodMNIST)

Data preprocessing: The class labels are converted to one hot encoding

CNN architecture:

Convolution layer, Activation,Pooling layer,
Convolution layer, Activation,Pooling layer,
Reshape layer,Dense layer,Sigmoid,Softmax

Experiments: For our experiments we will vary the Activa-

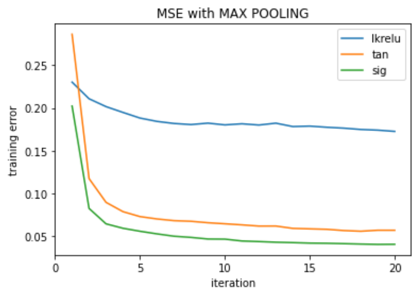

Fig. 28: CNN PneumoniaMNIST

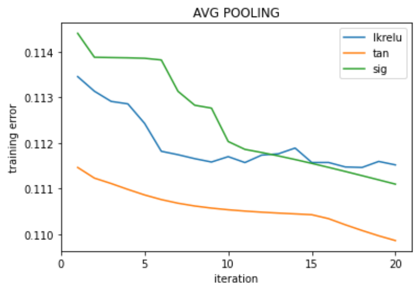

Fig. 29: CNN BloodMNIST

tion over Tanh, Leaky ReLU and Sigmoid functions, and vary the Pooling layer over Max pool and Average pool.
We have used the mean squared error loss function. The kernel size is 3x3 and the stride is 1. The learning rate is varied over [0.01, 0.03, 0.001, 0.003] and the model was trained for 20 iterations.

Following plots(Fig: 29,30) show the training error for CNN with different pooling layer against the number of iterations.

Observations: From the table(Table. XVI) we can observe that CNN with Sigmoid as activation function gives the best accuracy. All other models have performed decently. The best F1 score is given by Leaky ReLU with Max pooling. The best AUC is given by Tanh with Max pooling.

C. Bounding box regression Problem(Traffic dataset)

Data preprocessing: The class labels are converted to one hot encoding
CNN architecture:
Convolution layer, Activation, Pooling layer,

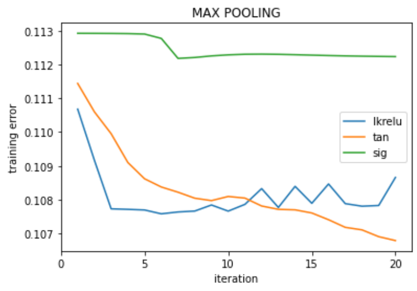

Fig. 30: CNN BloodMNIST

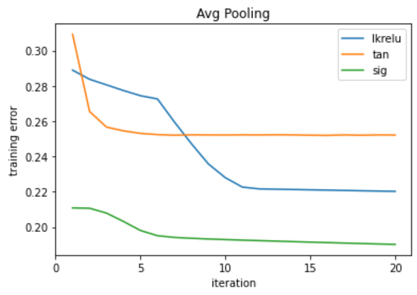

Fig. 31: CNN Traffic dataset

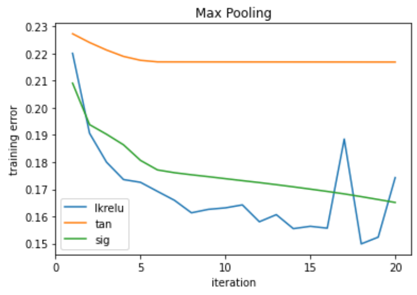

Fig. 32: CNN Traffic dataset

Convolution layer, Activation, Pooling layer,

Reshape layer, Dense layer, Sigmoid,

Experiments: For our experiments we will vary the Activation over Tanh, Leaky ReLU and Sigmoid functions, and vary the Pooling layer over Max pool and Average pool. We have used the mean squared error loss function. The kernel size is 10x10 and the stride is 1. The learning rate is varied over [0.01, 0.03, 0.001, 0.003]

Following plots(Fig: 31,32) show the training error for CNN with different pooling layer against the number of iterations.

Observations: From Table. XVII we can see that Leaky ReLU with Maxpool gave the least Mean square error and the mean absolute error and the highest mIoU.

D. Challenges and Learnings

- In order to efficiently code the Multilayer perceptron and Convolutional Neural Network and add batch norm layer, dropout layer etc it was imperative to adopt a modular coding style and add layers on top of each other each having a similar API.
- Using VGG 19 and Alex Net on small images (28*28*3) the image was getting reduced to size 0 while going through the network.
- Random crop and random rotation of the images in Question 3 acted as a very good regularizer and prevented overfitting.
- While coding CNN, the loss was decreasing very slowly or sometimes not at all. The problem was dying ReLU problem. So, we had to use leaky ReLU as the activation function which fixed the problem.

VI. Standard CNNs

A. Binary Classification problem(PneumoniaMNIST)

We applied Resnet50 and VGG19 in this question. We modified the last fully connected layer of each model to output 2 classes. We also modified the first layer to take Grayscale images.(1 input channel)

Observations and Conclusions: From the training plots (Fig. 33,34,35,36) we see that Resnet50 performs better than VGG19 and hence is a better model than VGG19. Also from the metrics plot, we see that accuracy(other metrics also) starts decreasing after the 6th epoch after which overfitting starts to take place. Therefore we should stop the training after the 6th epoch.

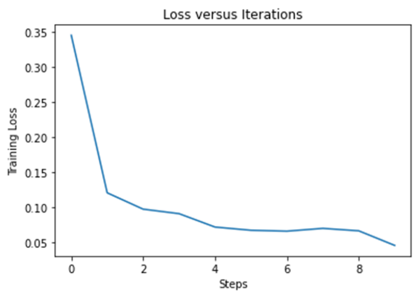

Fig. 33: VGG19 PneumoniaMNIST

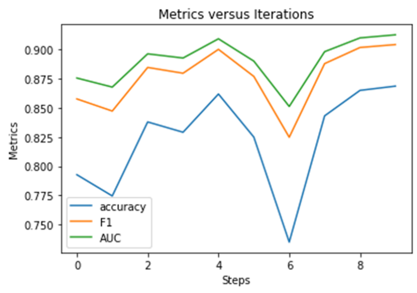

Fig. 34: VGG19 PneumoniaMNIST

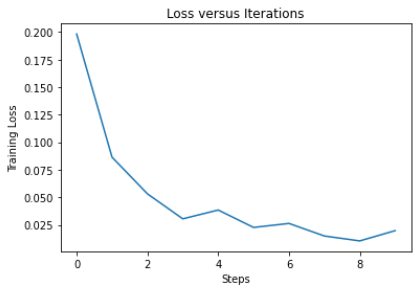

Fig. 35: Resnet50 PneumoniaMNIST

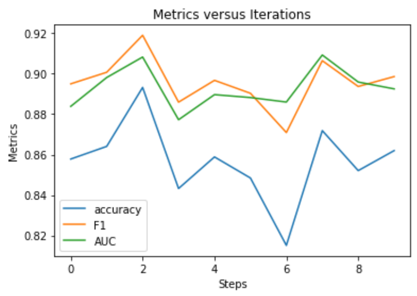

Fig. 36: Resnet50 PneumoniaMNIST

B. Multi-Class Classification Problem (BloodMNIST)

We tried Resnet 50 model architectures on this dataset. We modified the last fully connected layer of each model to output 8 classes. We also tried VGG 19 and AlexNet architectures as suggested but they don't take very small sized images. The image size was becoming 0 on down sampling through the VGG 19 and AlexNet architectures.

Observations and Conclusions: From the training plots (Fig. 37,38)we see that Resnet 50 performs better than the Multi layer Perceptron architectures. Also from the metrics plot of Resnet50, we see that accuracy(other metrics also) starts decreasing after the 7th epoch after which overfitting starts to take place. Therefore we should stop the training after the 7th epoch.

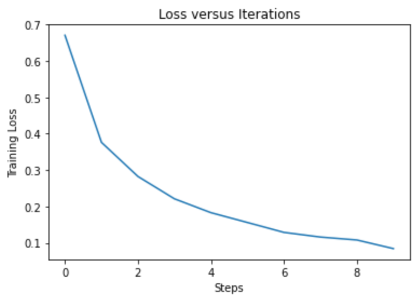

Fig. 37: Resnet50 BloodMNIST

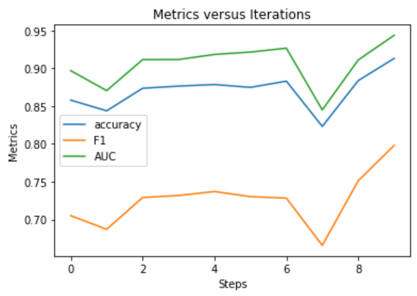

Fig. 38: Resnet50 BloodMNIST

C. Bounding box regression Problem(Traffic dataset)

Observations and Conclusions: The mIoU for Resnet34 was 0.526 and for VGG16 was 0.432 From both the plots (Fig. 39,40,41,42)we can see that Resnet provides better results as it's loss is less when compared to VGG 19 and also it's mean Intersection over Union score is higher. Overfitting starts to happen in both models after 5th iteration and so we should stop training after 5th iteration

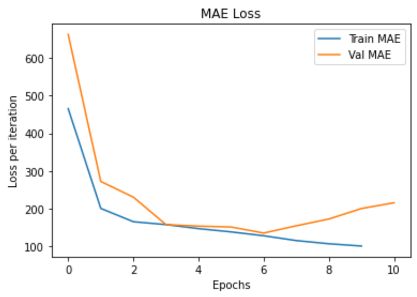

Fig. 39: VGG19 Traffic dataset

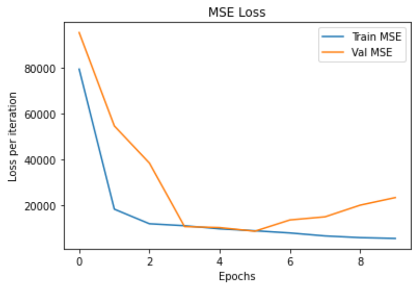

Fig. 40: VGG19 Traffic dataset

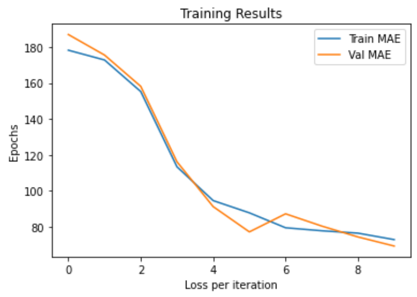

Fig. 41: Resnet50 Traffic dataset

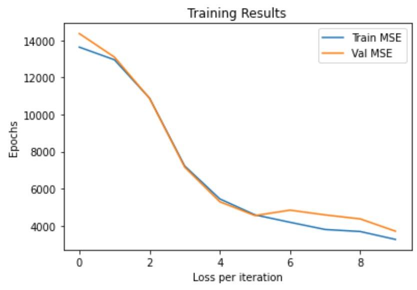

Fig. 42: Resnet50 Traffic dataset

VII. LSTM

LSTM NN architecture:
LSTM → Dropout → Dense layer(ReLU) → Dense layer(ReLU) → Dropout → Dense layer(ReLU) → Dropout → Dense layer(softmax)

Observations and Conclusions: From the training plots we can see that validation and training losses are roughly the

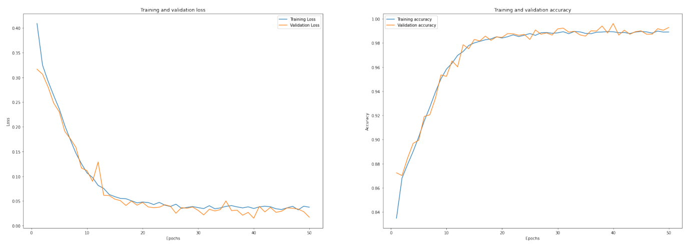

Fig. 43: LSTM: Loss and Accuracy Graph (batch size 16)

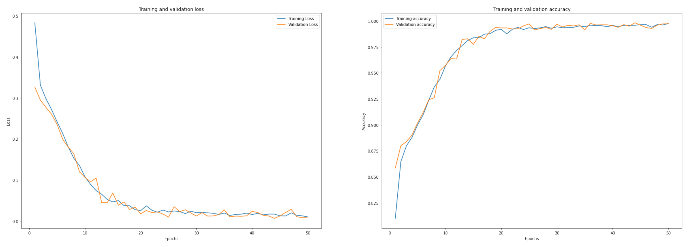

Fig. 44: LSTM : Loss and Accuracy Graph (batch size 64)

same which indicates a very low variance.Also the bias is very low indicating that this model works very good on the given dataset. We observed that True positive/false positive rates do not change substantially when comparing results after 10 epoch and 50 epoch(Table. XVIII).
True Positive/false positive rates changes slightly when we change batch size from 16 to 64.

TABLE XVIII: LSTM Metrics

		Avg. True Positive	Avg. False Positive	Avg. True Negative	Avg. False Negative
Batch size 16	10 epoch	0.270	0.082	0.574	0.073
	50 epoch	0.279	0.072	0.557	0.089
Batch size 64	10 epoch	0.274	0.078	0.561	0.086
	50 epoch	0.270	0.082	0.565	0.082