DEEP LEARNING ASSIGNMENT-1 REPORT

Part-1 SVM and Softmax

Observations

- For CIFAR-Dataset because of a lot of data variations the accuracy on CIFAR dataset is less compare to Mushroom dataset
- As CIFAR dataset is a large dataset, the model took more time to train.
- Softmax tries to minimize the error as much as possible but where as in svm we use hinge loss which stops training when the error is zero. So in case of cifar dataset softmax overfits so the accuracy is even worse.

Results

Classifier	CIFAR-Test	Mushroom-Test
SVM	22.70	79.38
Softmax	10.00	83.50

Part-2 Multilayer Classification

Model used :

Multi-layer fully connected neural network

Dimensions:

Input - `N X D` where N is the number of images

`Hi` : The number of neurons in the ith hidden layer

`Wi : H[i-1] X H[i]` : ith layer weights

`bi : H[i] X 1` : ith layer biases

`Zi  : H[i] X N` : Linear output of the ith layer

`Ai : H[i] X N` : Output after activation in the ith layer

Classification performed over C classes

Structure of the model:

L fully connected layers
ReLu Activation function for the first L-1 layers (to introduce non-linearity into the network)
Softmax activation for the last layer, which outputs the scores for each class

Input -> [Linear -> ReLu](L-1 times) -> LINEAR -> SOFTMAX -> Output

Loss function used:

Cross entropy loss with L2 regularization

Implementation:

Initializing the parameters

- Initializing the parameters (weights and biases) Wi and bi for the L layers

Training the model

- Forward propagation module
  - Linear output Z followed by ReLu activation function for the first L-1 layers
  - Linear output Z followed by Softmax activation function for the last layer, which gives the scores for each class for all data samples as a matrix scores where scores[i, c] is the score for class c on the input X[i]
- Computing the loss function at the final layer
- Backward propagation module :
  - Compute the gradients for the Wi parameters at each layer
  - Update the parameters with the gradients obtained using SGD

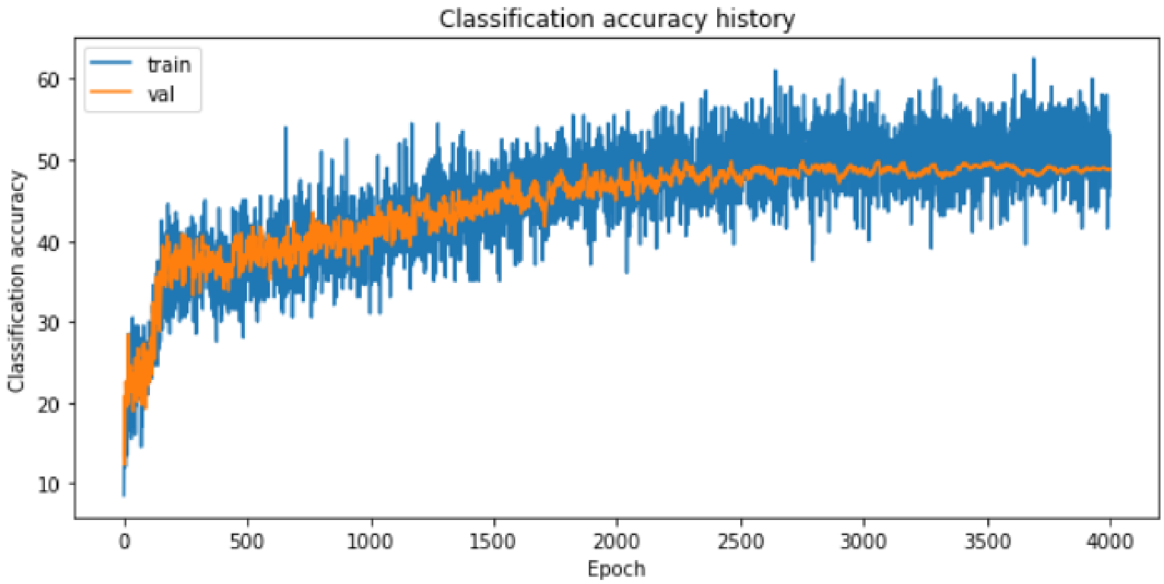
Testing

- With the trained parameters obtained, make a forward pass for the test set to obtain the output scores for each class and make appropriate predictions.

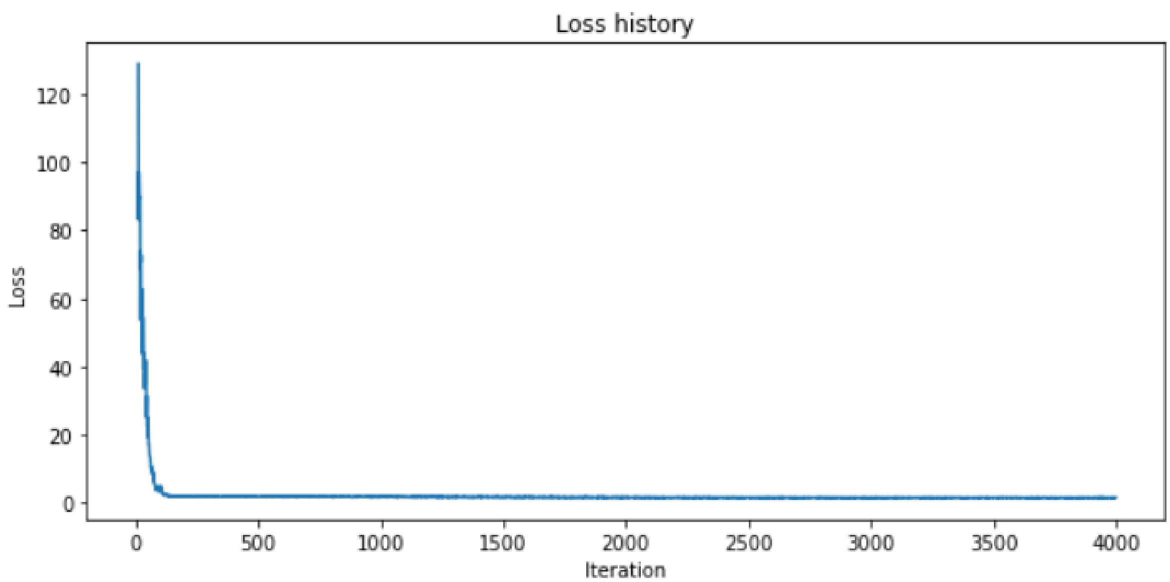
Observations and Results:

Accuracy on the test set obtained for SGD optimizer : 49.19%

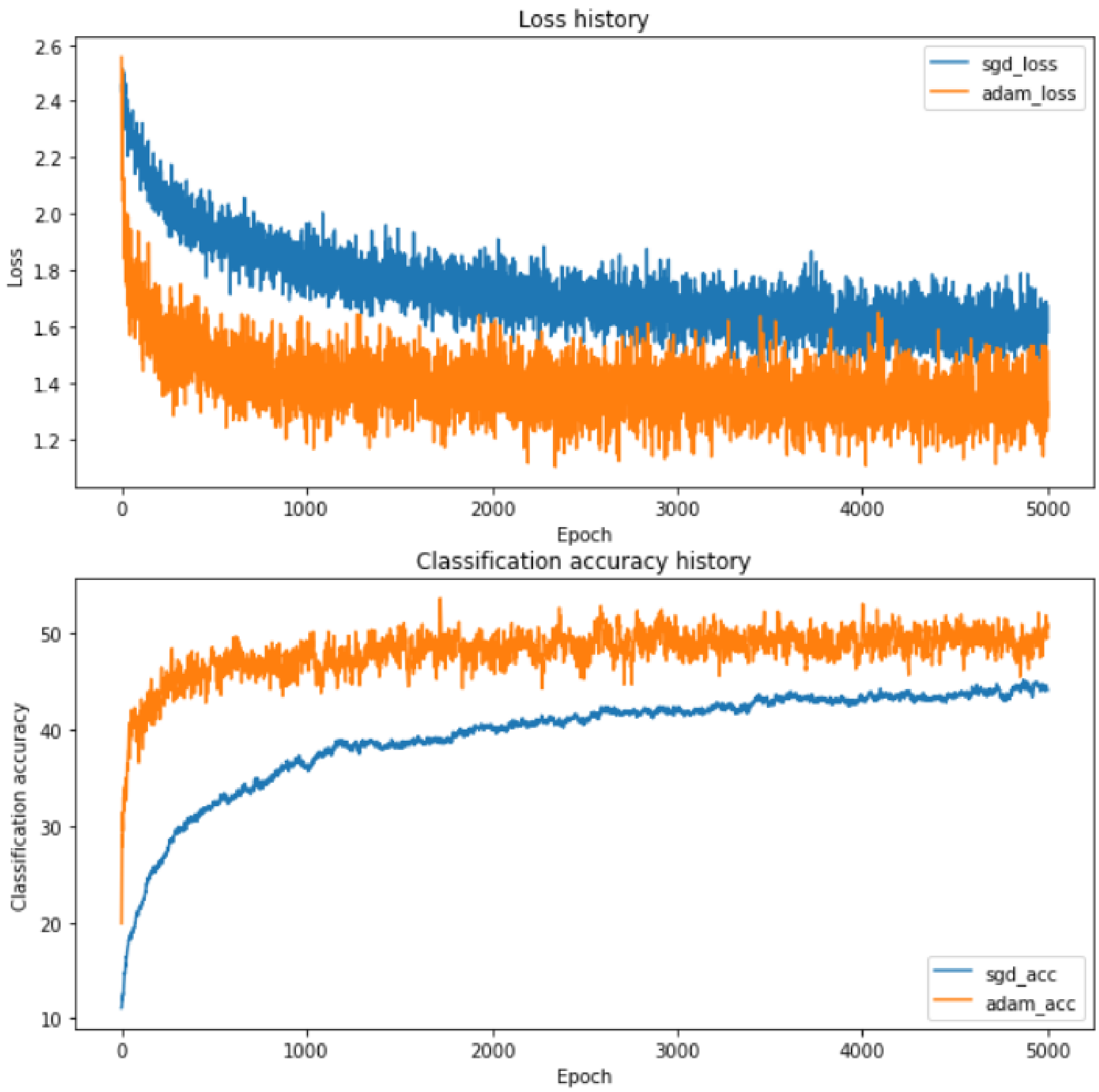
Variation of Classification accuracy with epochs for SGD



Variation of loss with iterations (SGD)



Comparing SGD optimizer with Adam Optimizer

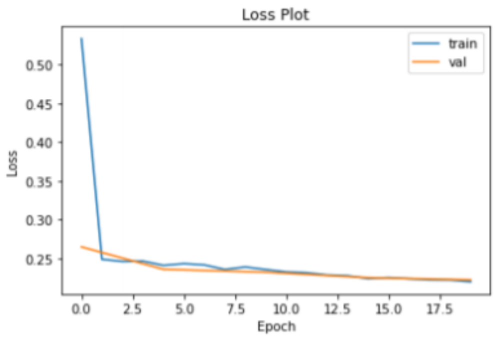


Part-3 Multi-label Image Classification

<u>**Part 3A - Predefined Models**</u>

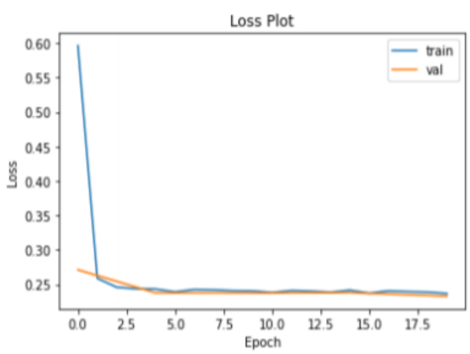
Observations

Simple classifier:



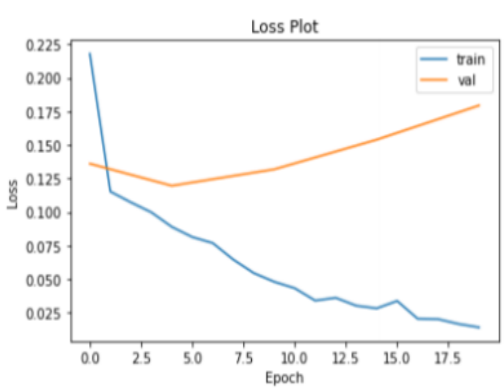
Here there is no gap between the train and validation error. This means we need to increase the capacity of the algorithm.

AlexNet :



Here there is no gap between the train and validation error. This means we need to increase the capacity of the algorithm.

Pretrained AlexNet :



Here the intersection of graphs says that the validation set is too small and statistics are not meaningful.

<u>Part 3B - Self models</u>

After studying VGG,AlexNet we tried some Conv layers and Fully connected layers we tried with different max pooling layers and COnv layers and Fully connected layers

```
class Classifier(nn.Module):
    class Classifier(nn.Module):
     # TODO: implement me
     def init(self):
         super(Classifier, self).init()
         self.features = nn.Sequential(nn.Conv2d(3, 64, 3),
                                       nn.Conv2d(64, 64, 3),
                                       nn.MaxPool2d(2),
                                       nn.Conv2d(64, 128, 3),
                                       nn.Conv2d(128, 128, 3),
                                       nn.MaxPool2d(2),
                                       nn.Conv2d(128, 256, 3),
                                       nn.Conv2d(256, 256, 3),
                                       nn.Conv2d(256, 256, 3),
                                       nn.MaxPool2d(2),
                                       nn.Conv2d(256, 512, 3),
                                       nn.Conv2d(512, 512, 3),
                                       nn.Conv2d(512, 512, 3),
                                       nn.MaxPool2d(2),
                                       nn.Conv2d(512, 512, 3),
                                       nn.Conv2d(512, 512, 3),
                                       nn.Conv2d(512, 512, 3),
                                       )
         self.para = nn.Sequential(
             nn.Dropout(),
             nn.ReLU(inplace=True),
             nn.Dropout(),
             nn.Linear(4096, 4096),
             nn.ReLU(inplace=True),
             nn.Linear(4096, NUM_CLASSES),
         )

     def forward(self, x):
         x = self.features(x)
         x = torch.flatten(x)
         x = self.para(x)
         return x
```

Individual Contributions

Korupolu Saideepthi - (S20180010087)
Part1 -Softmax
Part3A - Predefined Models

Varakala Sowmya - (S201800100187)
Part1- SVM
Part2 -Multilayer Classification -ForwardPass

ManjuShree - (S20180010055)
Part3A -Predefined Models
Part3B - Developing Your Own Classifier

Swathi k - (S20180010172)
Part2 - Multilayer Classification - BackwardPass
Part3B - Developing Your Own Classifier