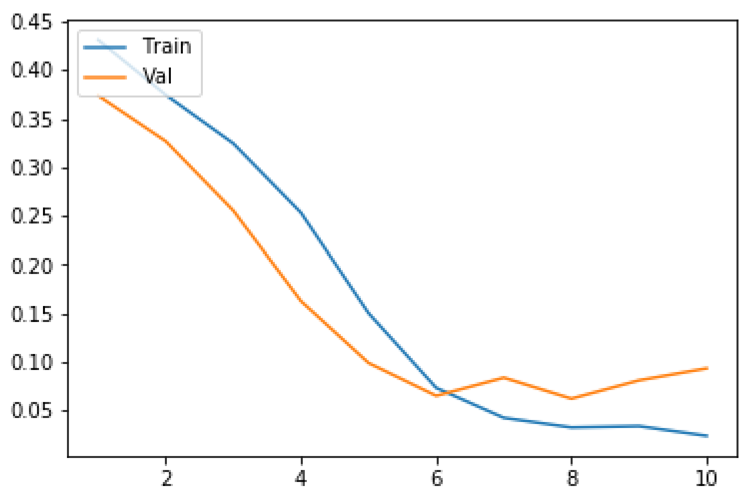
CV Assignment 5 Report

Roll: 2019121004
Name: Avani Gupta

1. Basic Network

```
MyNetwork(
  (model): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(4, 4), stride=(4, 4), padding=0, dilation=1,
ceil_mode=False)
    (3): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (4): ReLU()
    (5): MaxPool2d(kernel_size=(4, 4), stride=(4, 4), padding=0, dilation=1,
ceil_mode=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=2304, out_features=1024, bias=True)
    (8): ReLU()
    (9): Linear(in_features=1024, out_features=62, bias=True)
  )
)
```

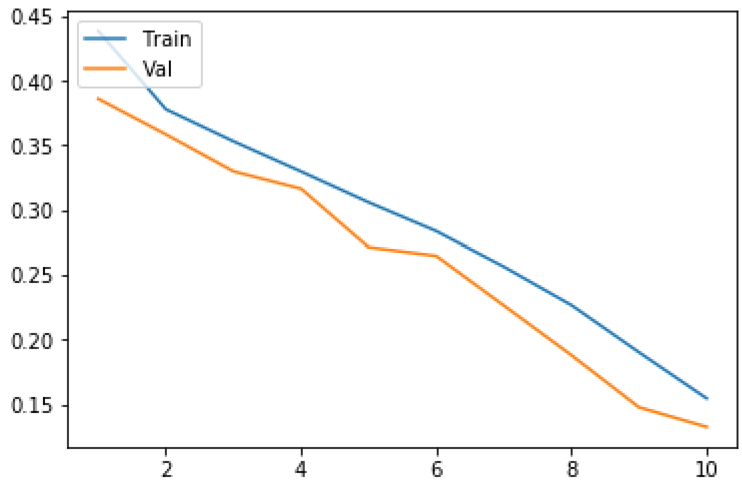
Loss vs Epochs



As we see the val loss starts to increase while train loss is still decreasing which suggests the model is overfitting.

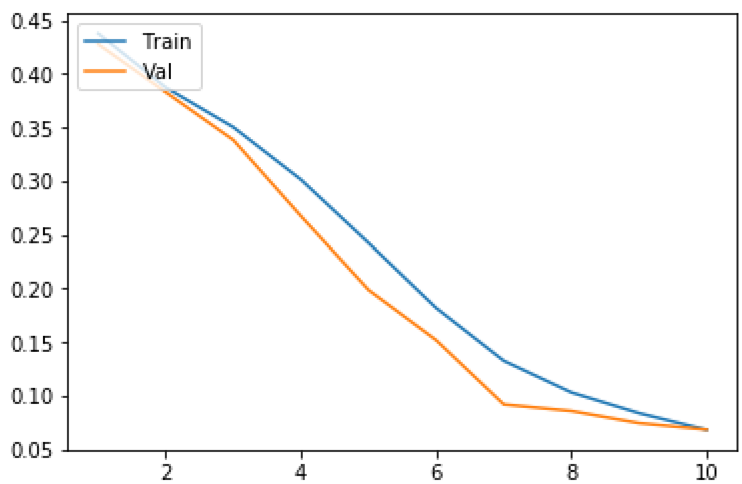
We will try various regularization techniques like BatchNorm and Dropout.

With BatchNorm



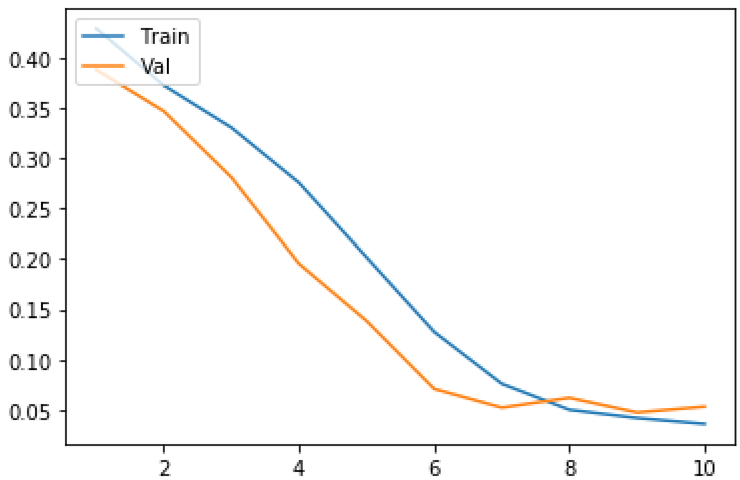
* Batch normalization regularizes the network which prevents overfitting.
* Helps in resolving vanishing gradient problem.
* Faster convergences and better performance.

With Dropout
* Regulation method.
* Reduces overfitting.
* Drop some of neurons randomly.
* Improved loss ias shown.



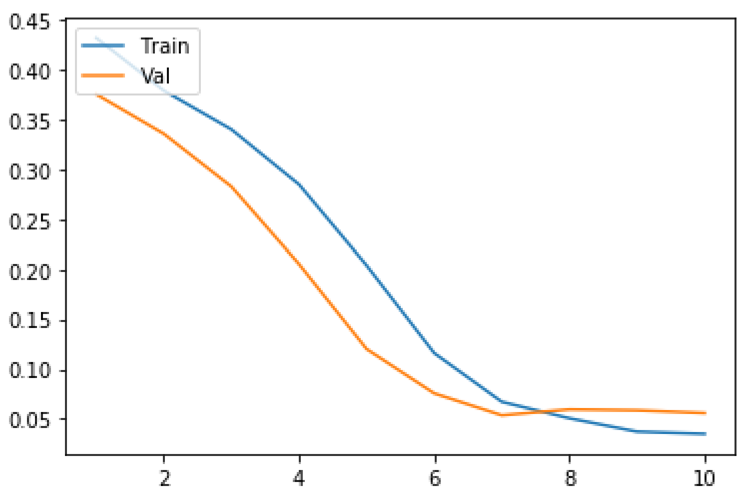
Playing with layers
MaxPool

* Max pooling is a sample-based discretization process.
* Used to reduce dimensionality of feature maps.
* Takes max operator over sliding window.



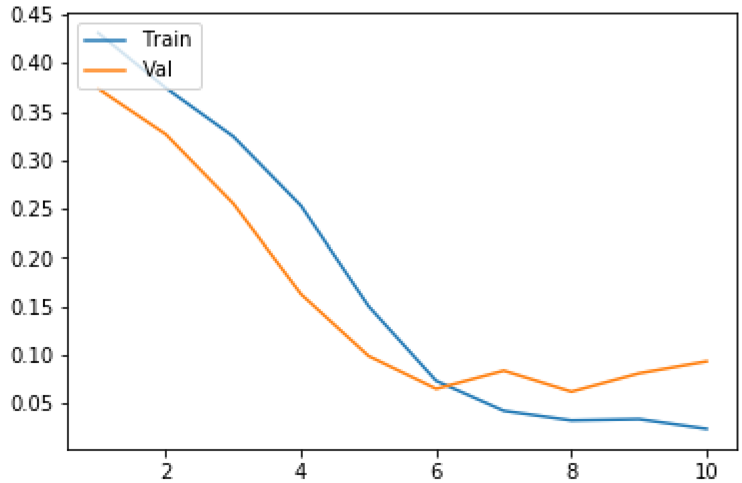
With Avg Pool

* Takes Avg operator over sliding window.



Playing with activation functions

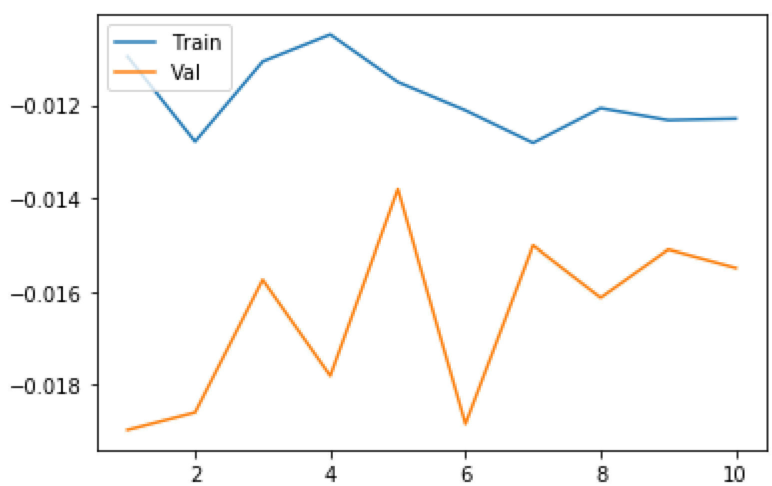
Our loss for basic model was nn.CrossEntropyLoss()
Which combines `LogSoftmax` and `NLLLoss`.



Now we change the NLLLoss which is Negative Loss Likelihood loss and add Softmax function at the end.

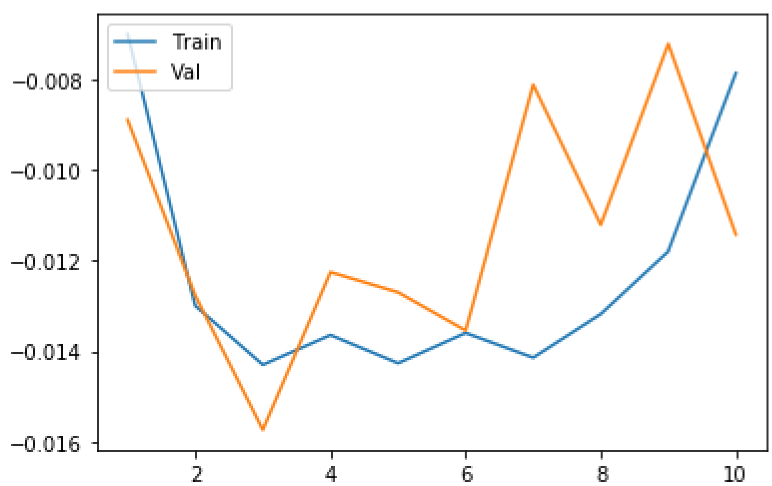
ReLU in between and Softmax at end

```
MyNetwork(
  (model): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=(4, 4), stride=(4, 4), padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (5): ReLU()
    (6): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): MaxPool2d(kernel_size=(4, 4), stride=(4, 4), padding=0, dilation=1, ceil_mode=False)
    (8): Flatten(start_dim=1, end_dim=-1)
    (9): Linear(in_features=2304, out_features=1024, bias=True)
    (10): ReLU()
    (11): Linear(in_features=1024, out_features=62, bias=True)
    (12): Softmax(dim=None)
  )
)
```



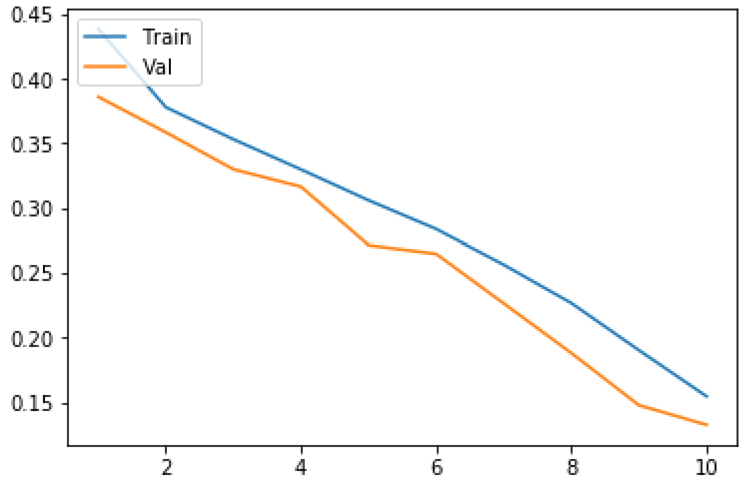
LeakyReLU in between and Softmax at end

```
MyNetwork(
  (model): Sequential(
    (0): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
    (1): LeakyReLU(negative_slope=0.01)
    (2): BatchNorm2d(6, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (3): MaxPool2d(kernel_size=(4, 4), stride=(4, 4), padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    (5): LeakyReLU(negative_slope=0.01)
    (6): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (7): MaxPool2d(kernel_size=(4, 4), stride=(4, 4), padding=0, dilation=1, ceil_mode=False)
    (8): Flatten(start_dim=1, end_dim=-1)
    (9): Linear(in_features=2304, out_features=1024, bias=True)
    (10): LeakyReLU(negative_slope=0.01)
    (11): Linear(in_features=1024, out_features=62, bias=True)
    (12): Softmax(dim=None)
  )
)
```



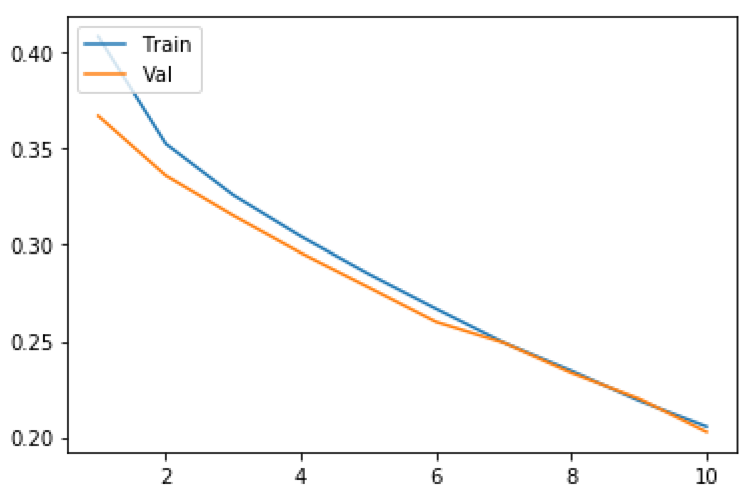
Different Optimizers
Adam

* Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.
* Adam is relatively easy to configure where the default configuration parameters do well on most problems.



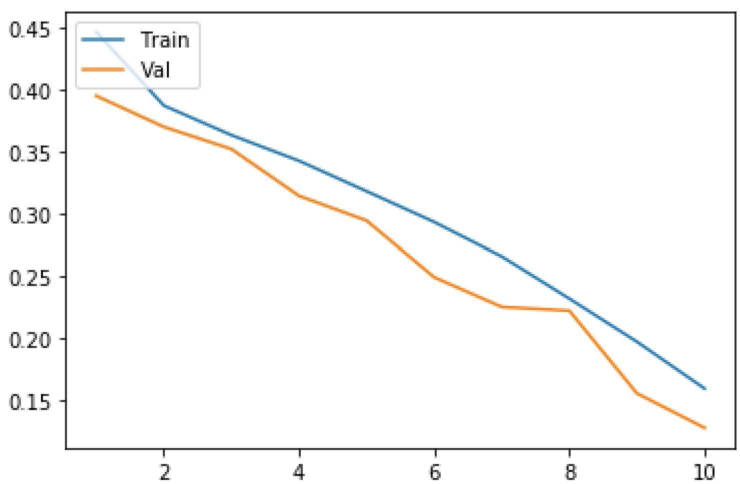
Adagrad

* For a sparse feature input where most of the values are zero, we can afford a higher learning rate which will boost the dying gradient resulting from these sparse features. If we have dense data, then we can have slower learning.
* Adjusts the learning rate according to values of gradient incurred.
* When larger updates take a smaller learning rate.
* When smaller updates increase learning rate.

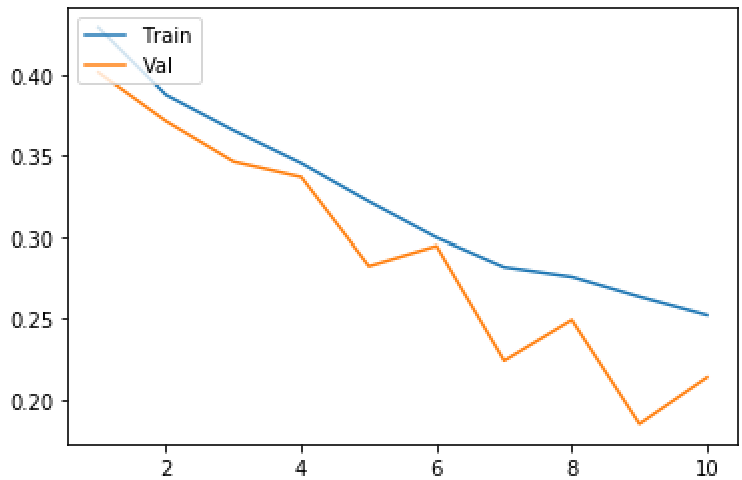


RMSProp

* Helps in vanishing gradient.
* Uses an adaptive learning rate
* Uses a moving average of squared gradients to normalize the gradient.
* This normalization balances the step size (momentum)
* Decreases the step for large gradients to avoid exploding
* Increases the step for small gradients to avoid vanishing.



SGD

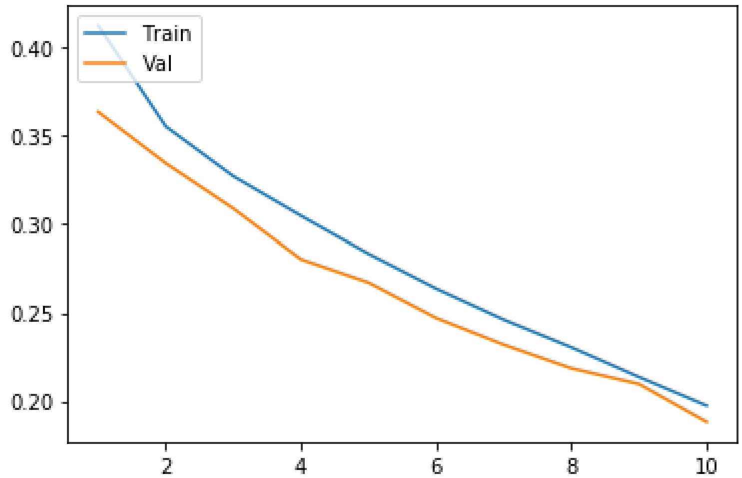


As expected SGD doesn't perform that well

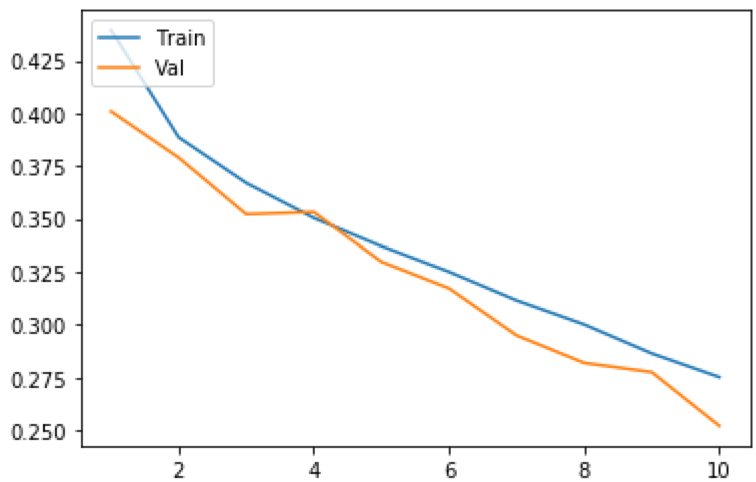
We observed the best convergence and model performance in Adam.

Augmentations

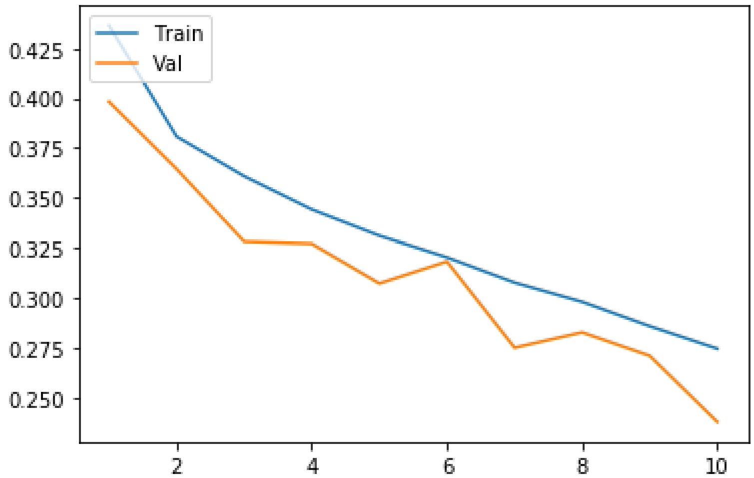
Without augmentations



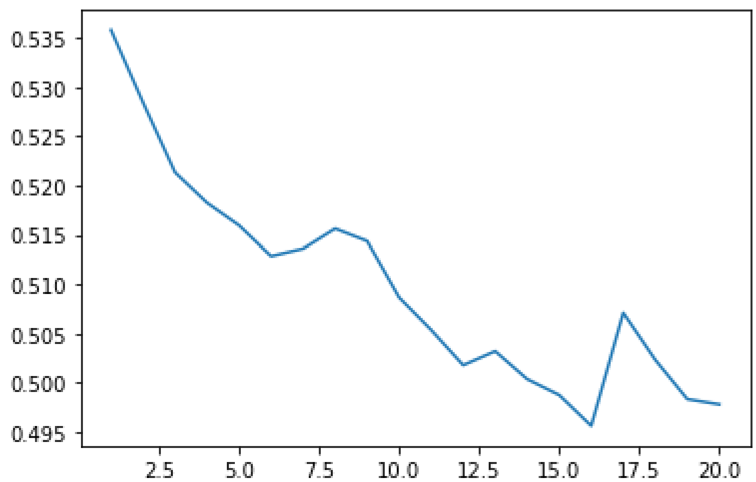
RandomHorizontalFlip(), colorJitter()



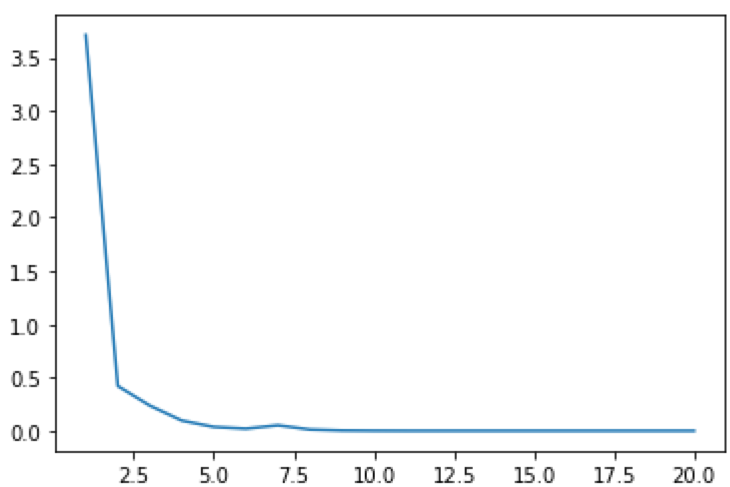
RandomHorizontalFlip()



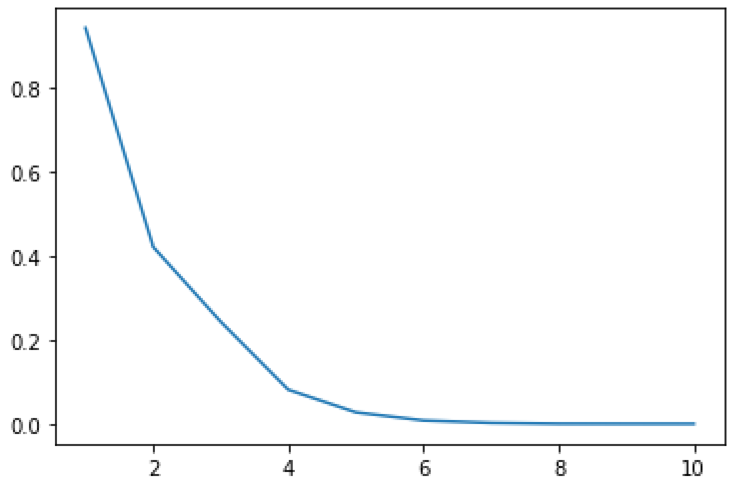
Part 1 on subset set of data
since training on entire dataset was time consuming I trained on subset of data(100 images just to study the relations between various parameters in depth)
Loss vs Epochs



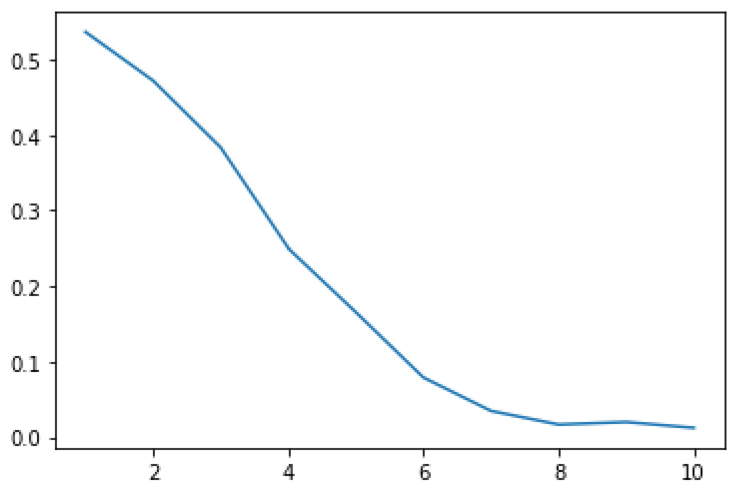
Without pooling in convnet
Features increase a lot
Loss vs Epochs



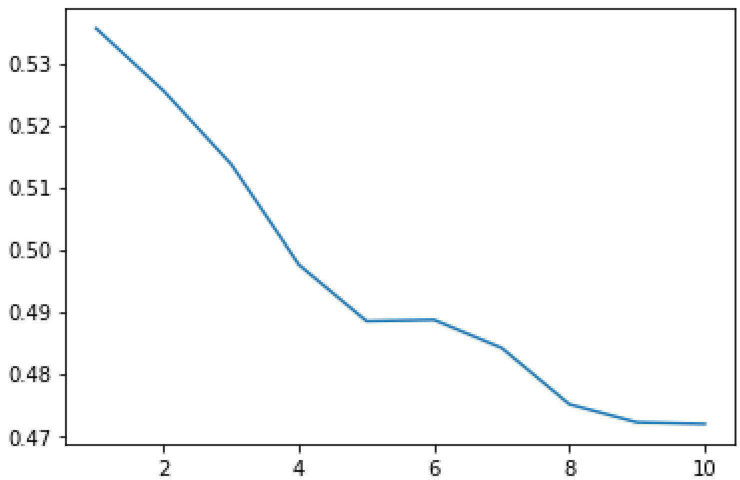
With MaxPool



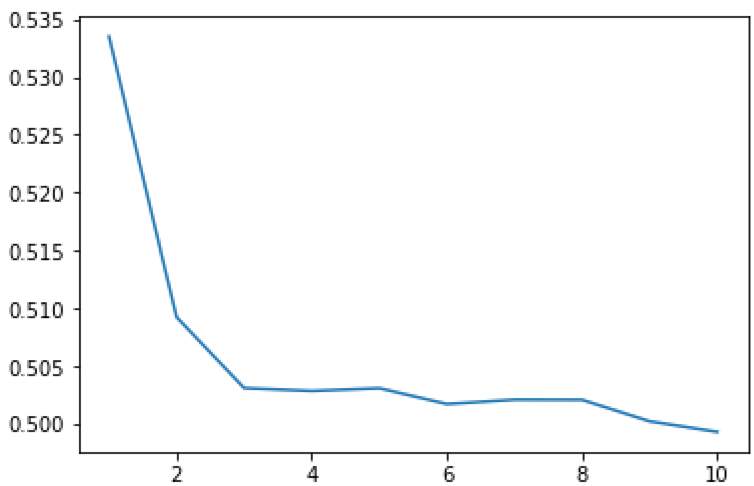
With Avg Pool



With Softmax at end

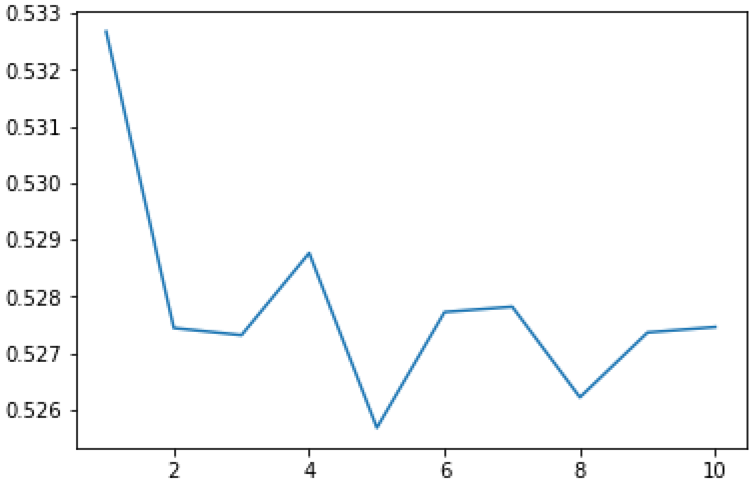


With Sigmoid at end

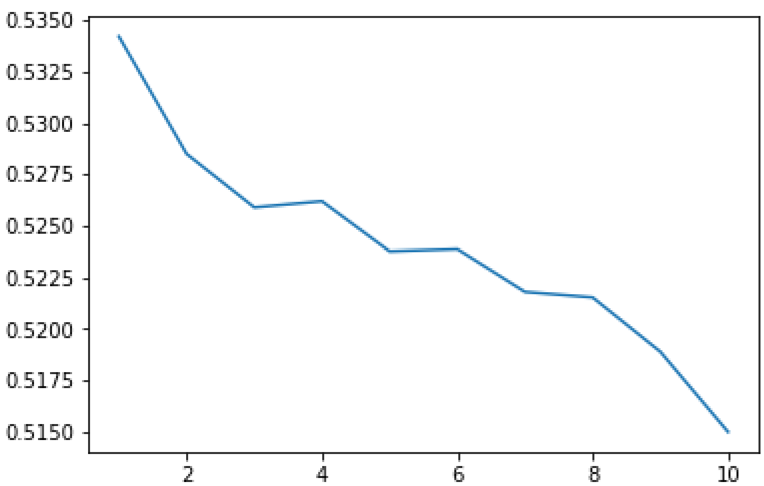


Different Optimizers

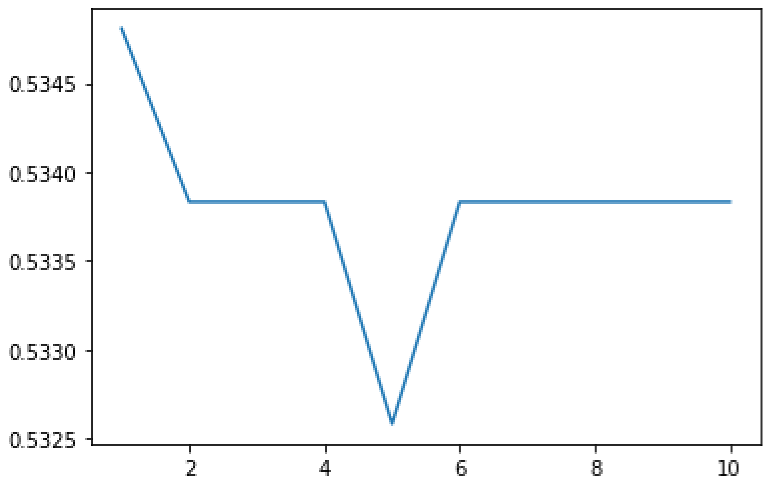
Adam



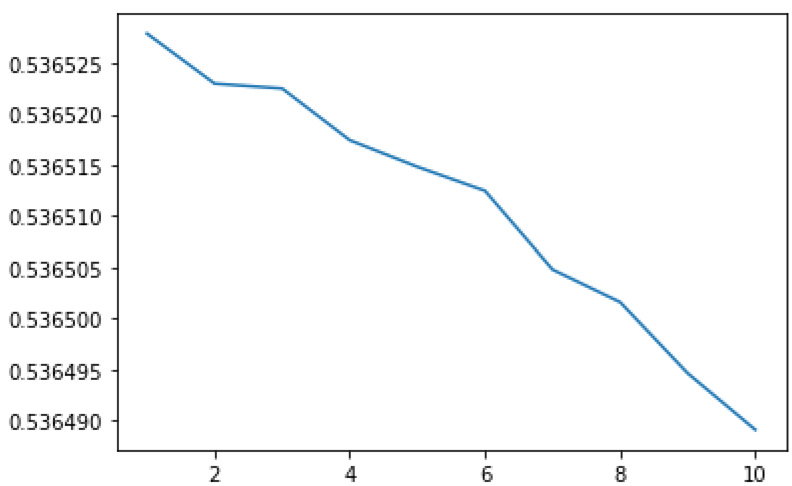
AdaGrad



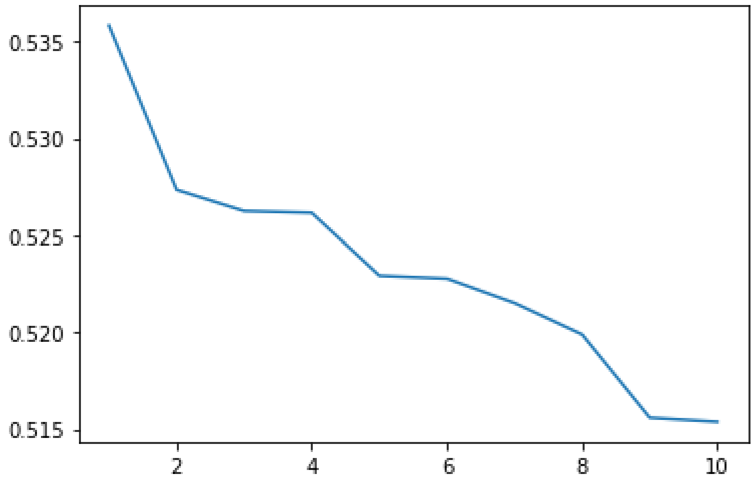
RMSProp


0.5345
0.5340
0.5335
0.5330
0.5325
2
4
6
8
10

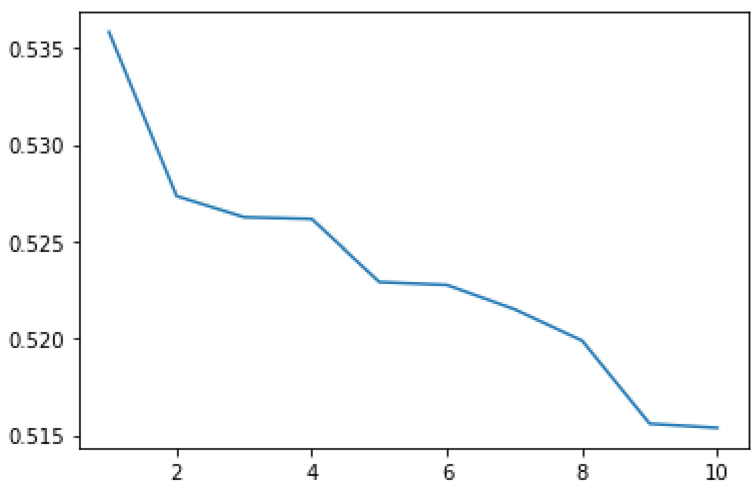
SGD


0.536525
0.536520
0.536515
0.536510
0.536505
0.536500
0.536495
0.536490
2
4
6
8
10

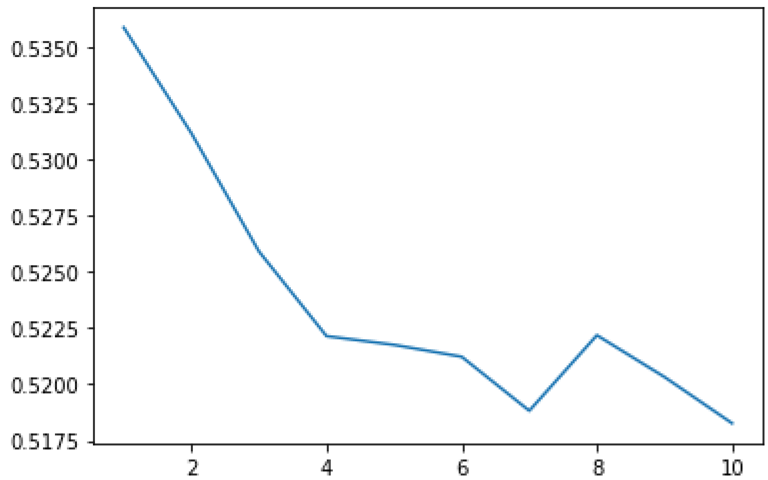
Data Augmentation
Without any augmentation



RandomFlip
Epoch: 10 | loss: 0.5075082015991211



Color Jitter
Epoch: 10 | loss: 0.5182729721069336



transforms.ColorJitter(),
transforms.RandomAffine(30, shear=0.2, scale = (0.8, 1.4)),
transforms.RandomGrayscale(p=0.9),
transforms.RandomPerspective()]), p=0.3)
Epoch: 10 | loss: 0.5175435400009155