PART a: CNN Classification of Ciphar10

Task : Classification of Ciphar10 using CNN

Steps used:

- Split data into train and test
- Collect data into batches (128 used here)
- Build a CNN model using pytorch
- Run these batches on a GPU using parallel processing (4 workers used)
- Check accuracy of model

Techniques/functions used and experimented with:

- ReLU activation
- Tanh activation
- Sigmoid activation
- Adam optimizer
- SGD without momentum
- SGD with momentum
- adaptive SGD with momentum (using learning scheduler)
- Adagrad optimizer
- Dropout (dropping a certain amount of nodes in the CNN)
- Cross_entropy loss

Some of the notable observations:

NOTE : Model1, Model2, Model 3 given below table

Model	Activation	Optimizer	Epochs	Other	Best Accuracy
Model1	ReLU	SGD	50	LR = 0.05	0.65
Model1	Tanh	SGD	50	LR = 0.05	0.62
Model1	ReLU	SGD	50	LR = 0.05, momentum =0.9,	0.71
Model1	ReLU	SGD	50	LR = 0.05, momentum =0.9, decay=0.9 gamma	0.76
Model1 with 40% drop	ReLU	SGD	50	LR = 0.05, momentum =0.9,	0.79

Model1	Tanh	SGD	50	LR = 0.05, momentum =0.9, decay=0.9 gamma	0.74
Model1	ReLU	Adam	30	LR = 0.001	0.759
Model1 with 40% drop	ReLU	Adam	50	LR = 0.001	0.82
Model1	Tanh	Adam	30	LR = 0.001	0.72
Model1	Sigmoid	Adam	50	LR = 0.05	0.12
Model1 with 40% drop	Tanh	Adam	50	LR = 0.001	0.77
Model2	Tanh	Adam	50	LR = 0.001	0.728
Model2	ReLU	Adam	50	LR = 0.001	0.73
Model3	ReLU	SGD	75	LR = 0.05, momentum =0.9, decay=0.9 gamma	0.79
Model3	Tanh	SGD	150	LR = 0.05, momentum =0.9,	0.75
Model3	Sigmoid	SGD	150	LR = 0.08, momentum =0.9,	0.11

Model 1: (activation of any kind used after each layer, 2 versions with/without dropout)

```
Conv2d(3, 32, kernel_size=3, padding=1),
Conv2d(32, 64, kernel_size=3, stride=1, padding=1)

MaxPool2d(2, 2), # output: 64 x 16 x 16
Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
MaxPool2d(2, 2), # output: 128 x 8 x 8

Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
Conv2d(256, 256, kernel_size=3, stride=1, padding=1),

MaxPool2d(2, 2), # output: 256 x 4 x 4
#Dropout(p=0.4),

Flatten(),
Linear(256*4*4, 1024),
Linear(1024, 512),
Linear(512, 10))
```

Model 2: (activation of any kind used after each layer)

```
Conv2d(3, 32, kernel_size=3, padding=1),
Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
MaxPool2d(2, 2), # output: 64 x 16 x 16
Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
MaxPool2d(2, 2), # output: 128 x 8 x 8
Dropout(0.4),
Flatten(),
Linear(128*8*8, 1024),
Dropout(0.2),
Linear(1024, 512),
Linear(512, 10))
```

Model 3: (activation of any kind used after each layer)

```
Conv2d(3, 32, kernel_size=3, padding=1),                ,
Conv2d(32, 64, kernel_size=3, stride=1, padding=1),          ,
MaxPool2d(2, 2), # output: 64 x 16 x 16

Conv2d(64, 128, kernel_size=3, stride=1, padding=1),         ,

MaxPool2d(2, 2), # output: 128 x 8 x 8
Dropout(0.4),

Flatten(),
Linear(128*8*8, 1024),          ,
Dropout(0.2),
Linear(1024, 512),          ,
Linear(512, 256),           ,
Linear(256, 10))
```

Observations and models used with graphs:

1) model = 6 Convolutional layers, 3 Linear layers (MODEL1)

Learning rate =0.04,

ReLU activation

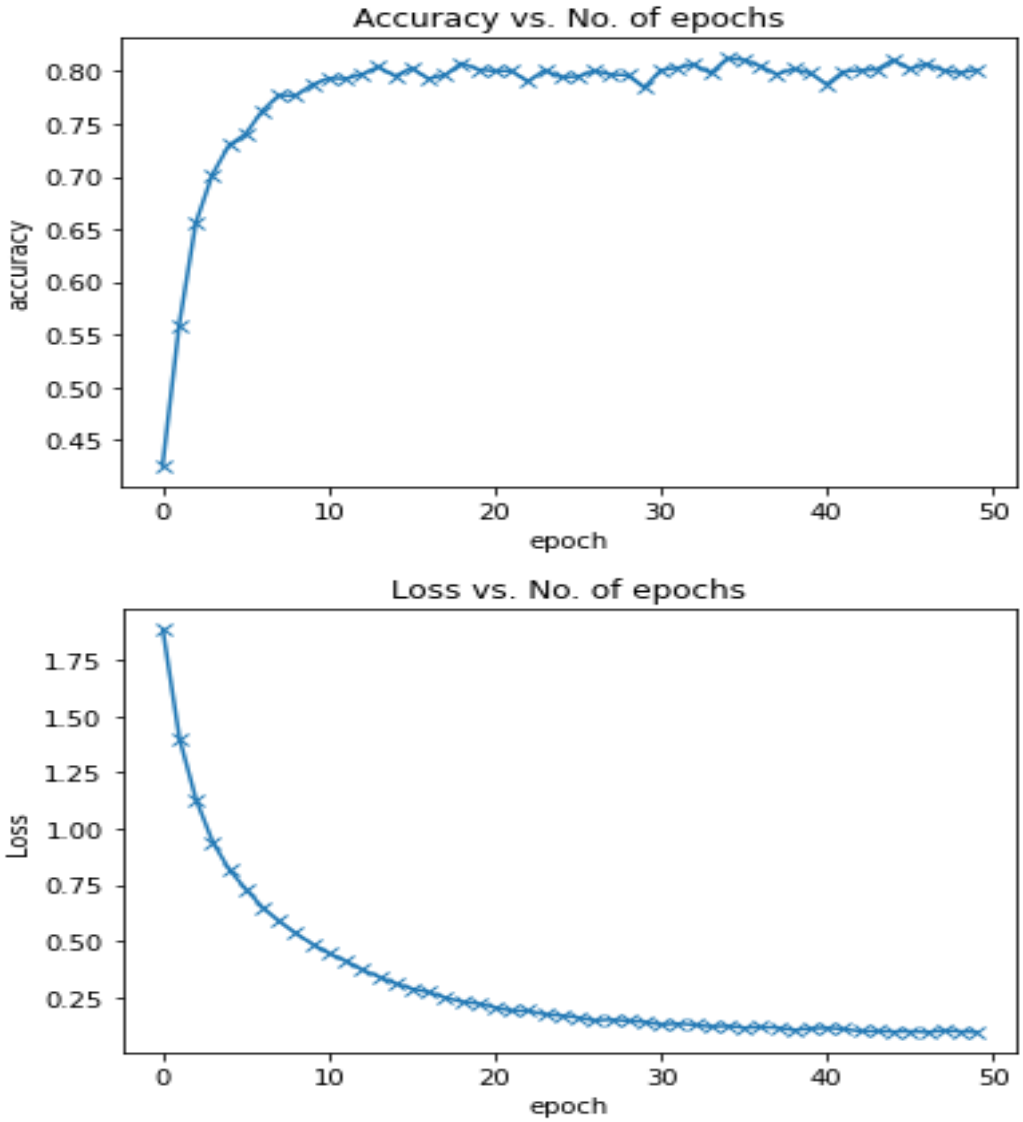
50 epochs,

Optimizer = SGD

momentum=0.9

Results from the last few epochs

```
Epoch [45], train_loss: 0.0857, val_loss: 1.8315, val_acc: 0.7054
Epoch [46], train_loss: 0.1098, val_loss: 1.5996, val_acc: 0.7170
Epoch [47], train_loss: 0.1078, val_loss: 1.7279, val_acc: 0.7232
Epoch [48], train_loss: 0.0996, val_loss: 1.8753, val_acc: 0.7136
Epoch [49], train_loss: 0.1246, val_loss: 1.8270, val_acc: 0.7111
```



2) model = 6 Convolutional layers, 3 Linear layers (MODEL1)

Learning rate =0.04,

ReLU activation

50 epochs,

Optimizer = SGD

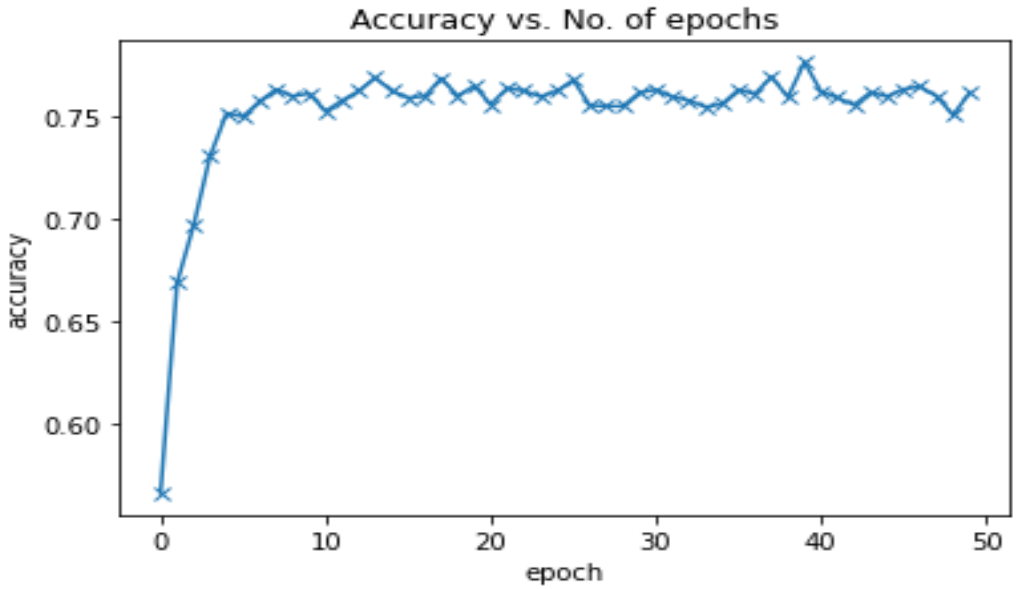
momentum=0.9

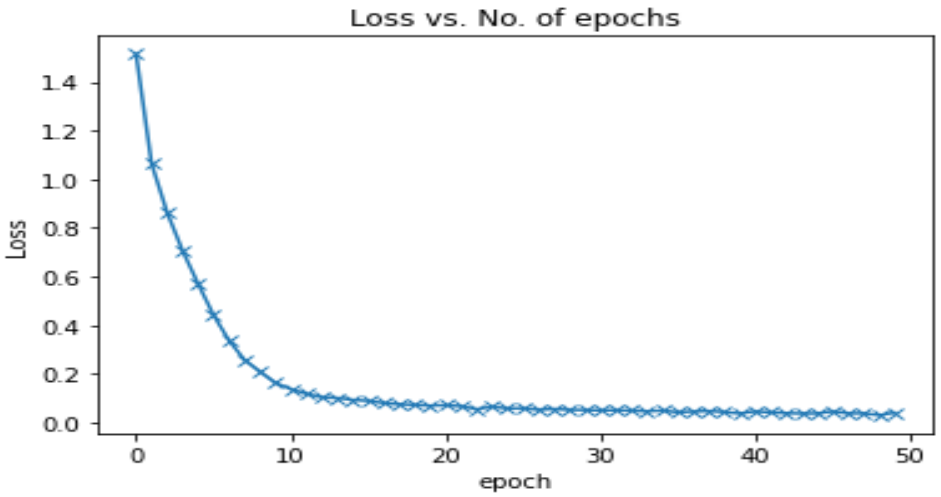
exponential decay = gamma =0.9

Results from the last few epochs

```
Epoch [47], train_loss: 0.0373, val_loss: 1.4779, val_acc: 0.7597
Epoch [48], train_loss: 0.0298, val_loss: 1.5027, val_acc: 0.7506
```

```
Epoch [49], train_loss: 0.0395, val_loss: 1.3936, val_acc: 0.7614
```





3) model = 6 Convolutional layers, 3 Linear layers (MODEL1)

Learning rate =0.05

Tanh activation

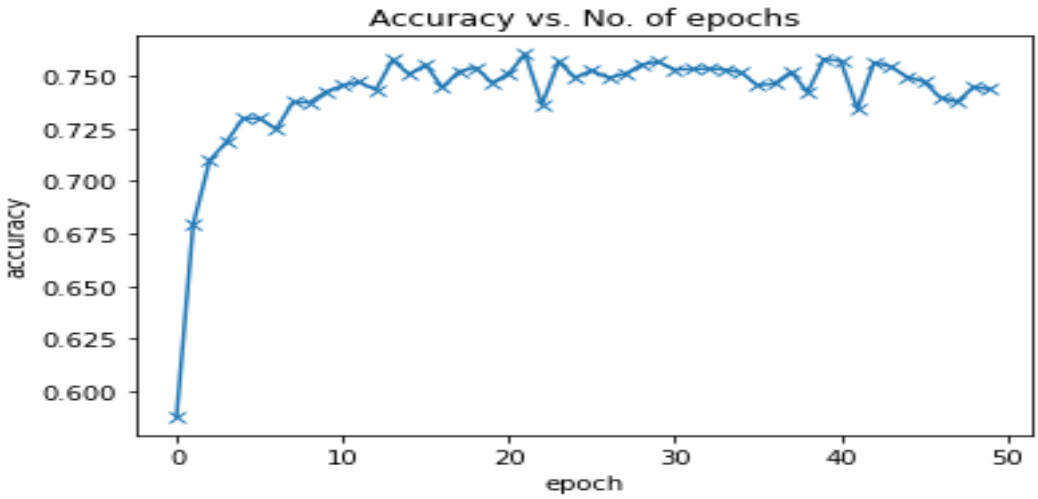
Momentum = 0.9

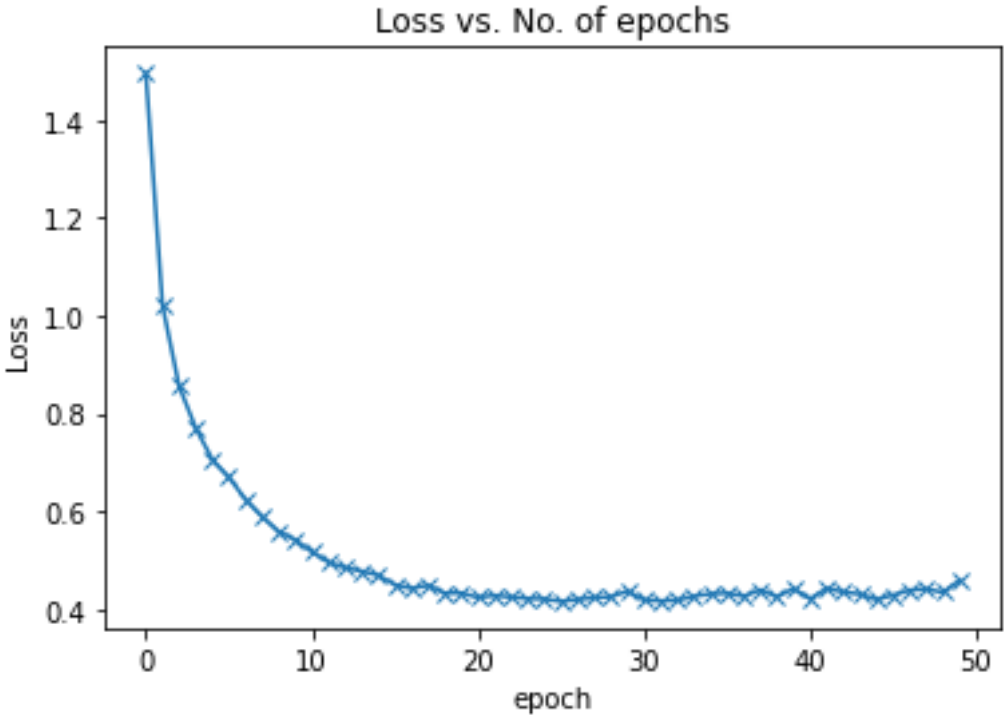
Learning scheduler decay = 0.9(gamma)

50 epochs

<u>Results from the last few epochs</u>

```
Epoch [45], train_loss: 0.4290, val_loss: 0.8198, val_acc: 0.7473
Epoch [46], train_loss: 0.4390, val_loss: 0.8301, val_acc: 0.7398
Epoch [47], train_loss: 0.4412, val_loss: 0.8416, val_acc: 0.7377
Epoch [48], train_loss: 0.4346, val_loss: 0.8145, val_acc: 0.7448
Epoch [49], train_loss: 0.4591, val_loss: 0.8362, val_acc: 0.7438
```





4) model = 6 Convolutional layers, 3 Linear layers with 40% dropout (MODEL1)

Learning rate =0.05

ReLU activation

Momentum = 0.9

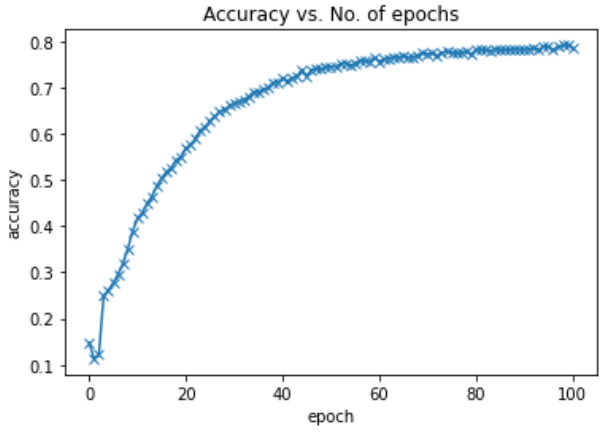
Learning scheduler decay = 0.9(gamma)

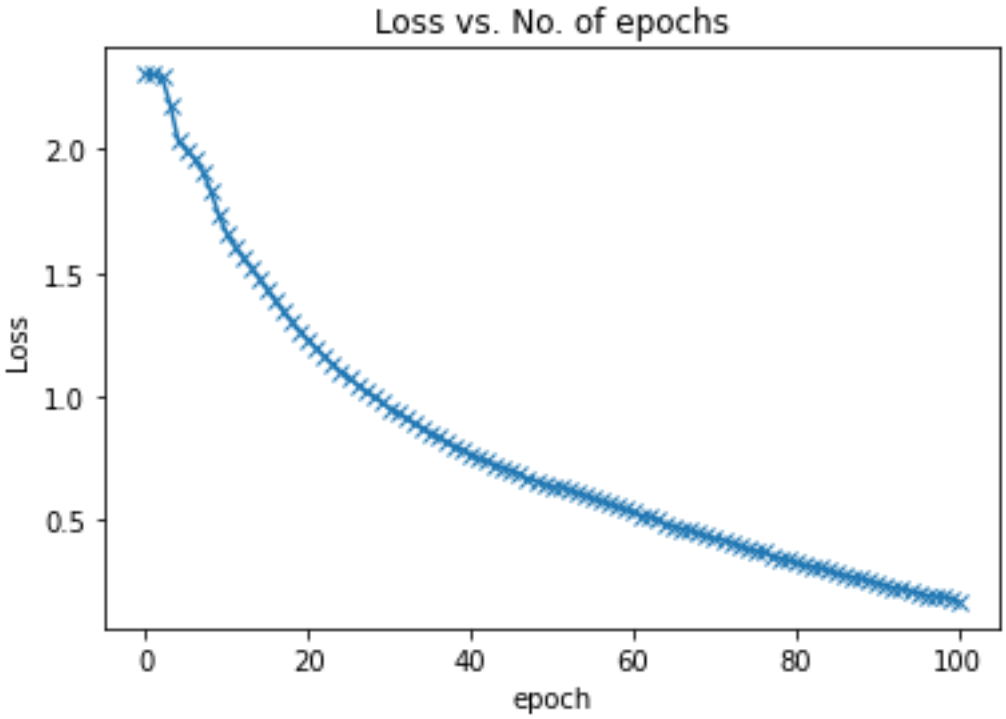
50 epochs

Results from the last few epochs

```
Epoch [46], train_loss: 0.1933, val_loss: 0.7606, val_acc: 0.7879
Epoch [47], train_loss: 0.1906, val_loss: 0.7763, val_acc: 0.7918
Epoch [48], train_loss: 0.1854, val_loss: 0.7635, val_acc: 0.7911
```

```
Epoch [49], train_loss: 0.1721, val_loss: 0.7886, val_acc: 0.7871
```





5) model = 6 Convolutional layers, 3 Linear layers (MODEL1)

Optimizer=Adam,

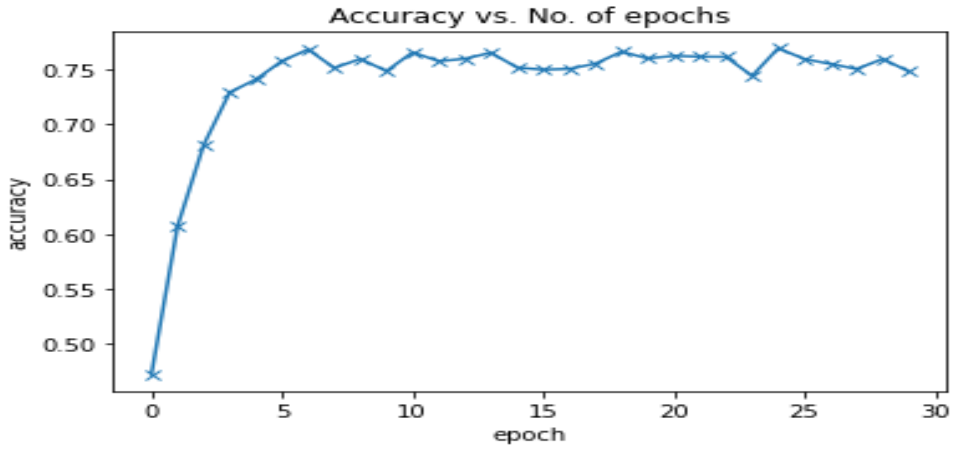
Learning rate = 0.001,

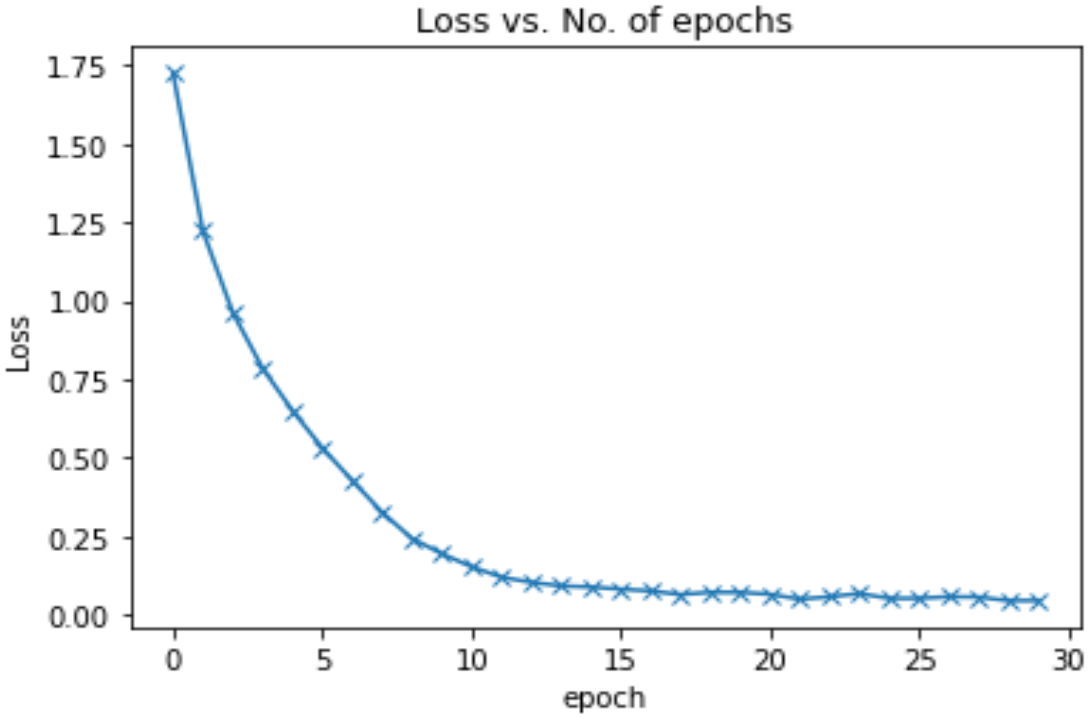
ReLU activation

30 epochs

<u>Results from the last few epochs</u>

```
Epoch [25], train_loss: 0.0516, val_loss: 1.6066, val_acc: 0.7594
Epoch [26], train_loss: 0.0568, val_loss: 1.5609, val_acc: 0.7549
Epoch [27], train_loss: 0.0557, val_loss: 1.5106, val_acc: 0.7503
Epoch [28], train_loss: 0.0442, val_loss: 1.6479, val_acc: 0.7594
Epoch [29], train_loss: 0.0445, val_loss: 1.7476, val_acc: 0.7480
```





6) model = 6 Convolutional layers, 3 Linear layers(MODEL1)

Learning rate =0.001,

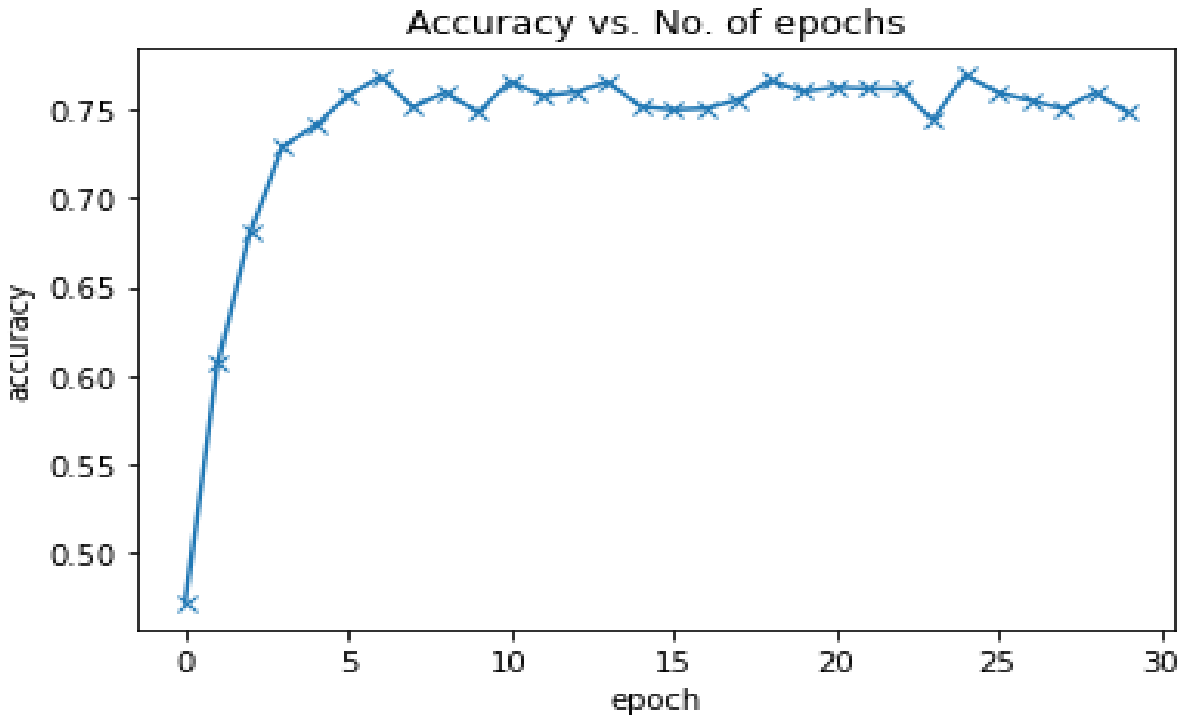
Optimizer = Adam

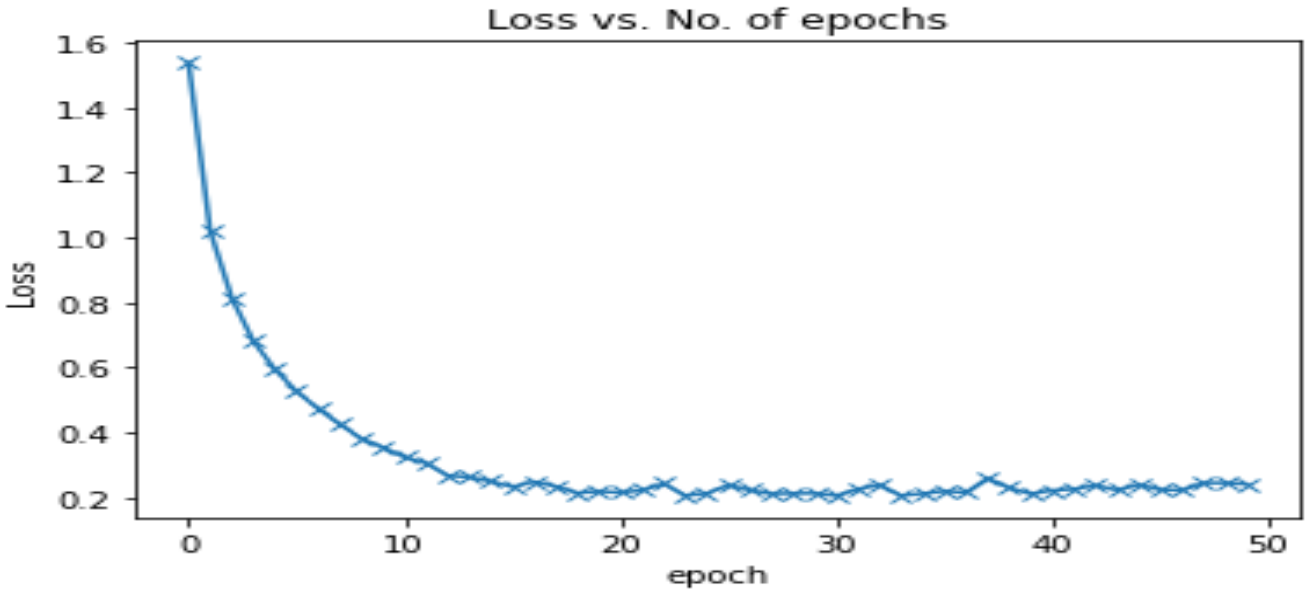
Tanh Activation

50 epochs

Results from the last few epochs

```
Epoch [45], train_loss: 0.2237, val_loss: 1.1024, val_acc: 0.7283
Epoch [46], train_loss: 0.2218, val_loss: 1.1290, val_acc: 0.7286
Epoch [47], train_loss: 0.2442, val_loss: 1.1320, val_acc: 0.7253
Epoch [48], train_loss: 0.2441, val_loss: 1.1074, val_acc: 0.7298
Epoch [49], train_loss: 0.2405, val_loss: 1.1308, val_acc: 0.7232
```





7) model = 6 Convolutional layers, 3 Linear layers, 40% dropout before the linear layers begin (MODEL1)

Learning rate =0.001

ReLU activation,

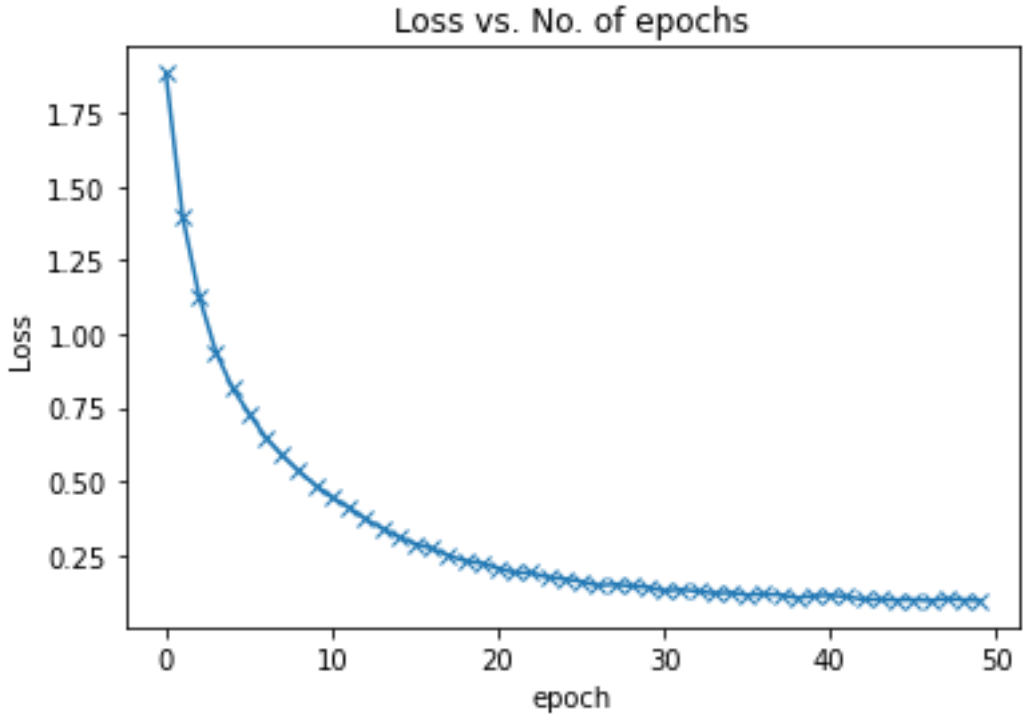
50 epochs

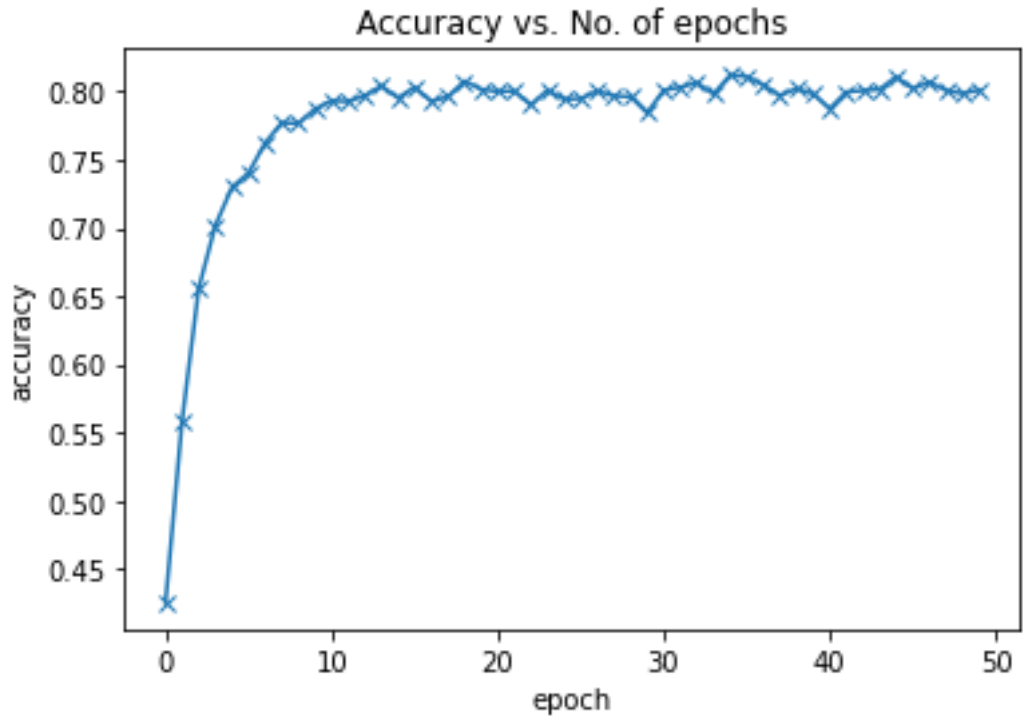
Optimizer = Adam

Results from the last few epochs

```
Epoch [44], train_loss: 0.0961, val_loss: 0.9191, val_acc: 0.8110
```

```
Epoch [45], train_loss: 0.0975, val_loss: 0.9570, val_acc: 0.8023
Epoch [46], train_loss: 0.0938, val_loss: 0.9723, val_acc: 0.8071
Epoch [47], train_loss: 0.1002, val_loss: 0.9361, val_acc: 0.8009
Epoch [48], train_loss: 0.0972, val_loss: 0.9869, val_acc: 0.7987
Epoch [49], train_loss: 0.0938, val_loss: 0.9710, val_acc: 0.8013
```





8) model = 6 Convolutional layers, 3 Linear layers (MODEL1)

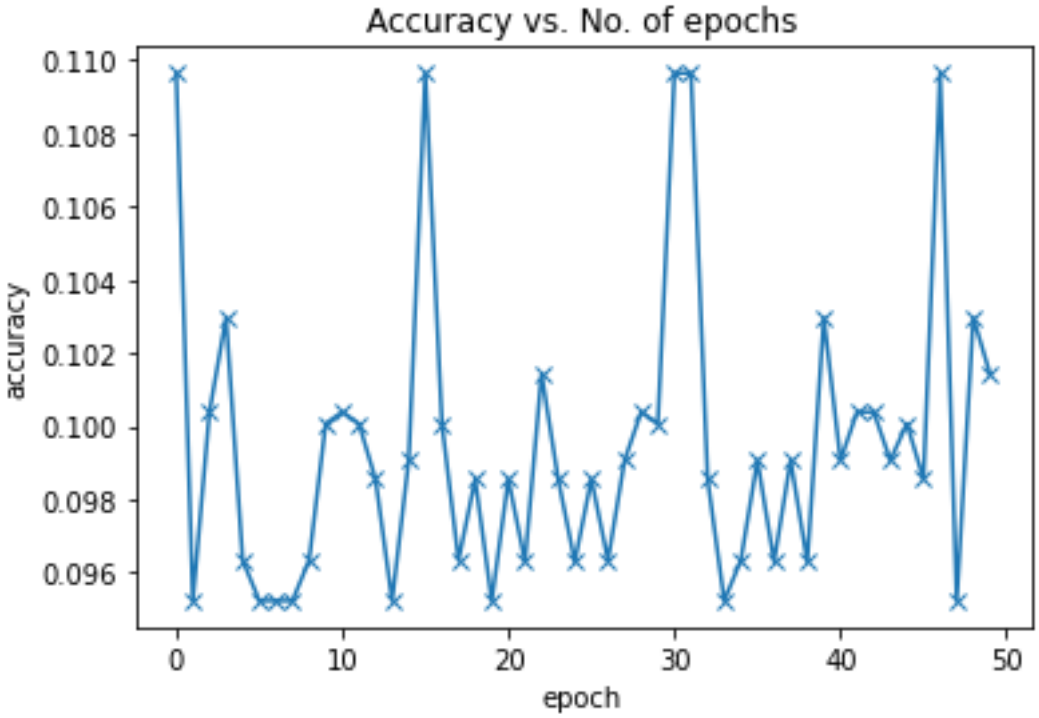
Sigmoid activation

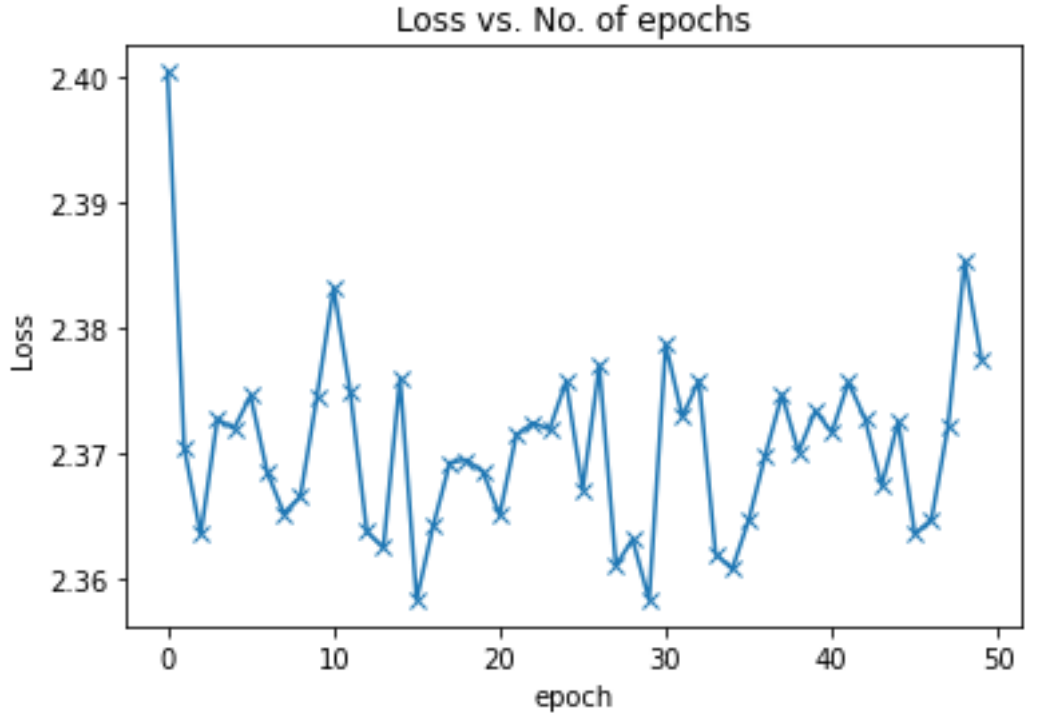
50 epochs

Learning rate =0.009

Optimizer = Adam

```
Epoch [45], train_loss: 2.3636, val_loss: 2.3306, val_acc: 0.0986
Epoch [46], train_loss: 2.3647, val_loss: 2.3448, val_acc: 0.1097
Epoch [47], train_loss: 2.3722, val_loss: 2.4105, val_acc: 0.0952
Epoch [48], train_loss: 2.3853, val_loss: 2.3602, val_acc: 0.1030
Epoch [49], train_loss: 2.3776, val_loss: 2.4068, val_acc: 0.10
```





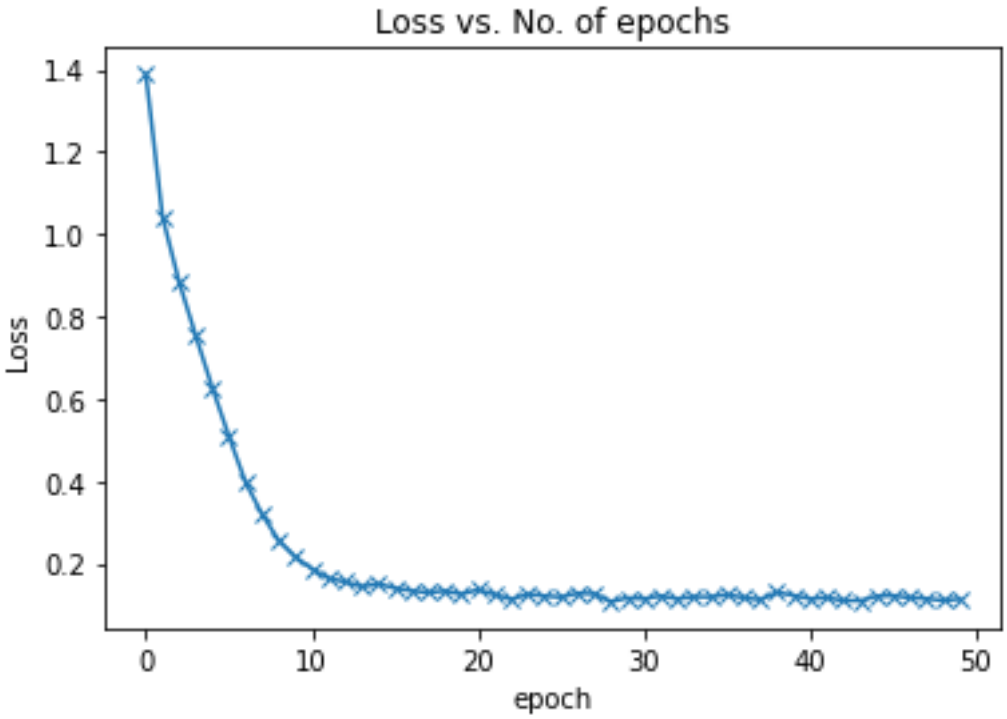
9) Model = 3 Convolutional layers, 3 Linear layers (MODEL2)

Optimizer= Adam

Learning rate = 0.001

Tanh activation

```
Epoch [45], train_loss: 0.1219, val_loss: 1.3690, val_acc: 0.7254
Epoch [46], train_loss: 0.1167, val_loss: 1.3279, val_acc: 0.7292
Epoch [47], train_loss: 0.1145, val_loss: 1.3286, val_acc: 0.7280
Epoch [48], train_loss: 0.1103, val_loss: 1.3724, val_acc: 0.7252
Epoch [49], train_loss: 0.1136, val_loss: 1.3734, val_acc: 0.7232
```

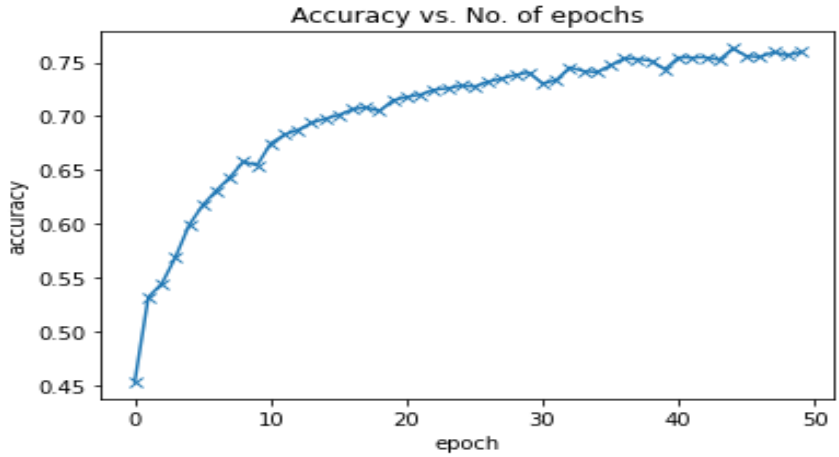


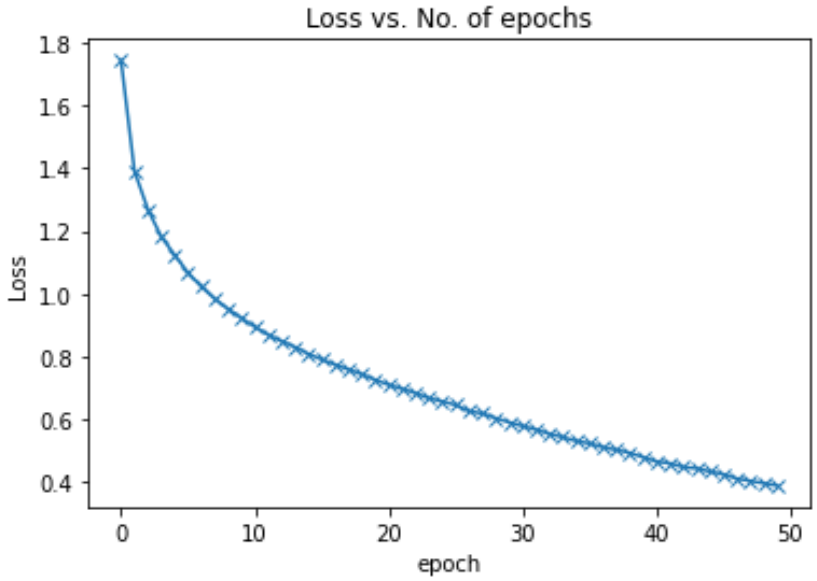
10) Model = 3 Convolutional layers, 3 Linear layers (MODEL2)

ReLU activation

Optimizer= Adam

Learning rate = 0.001





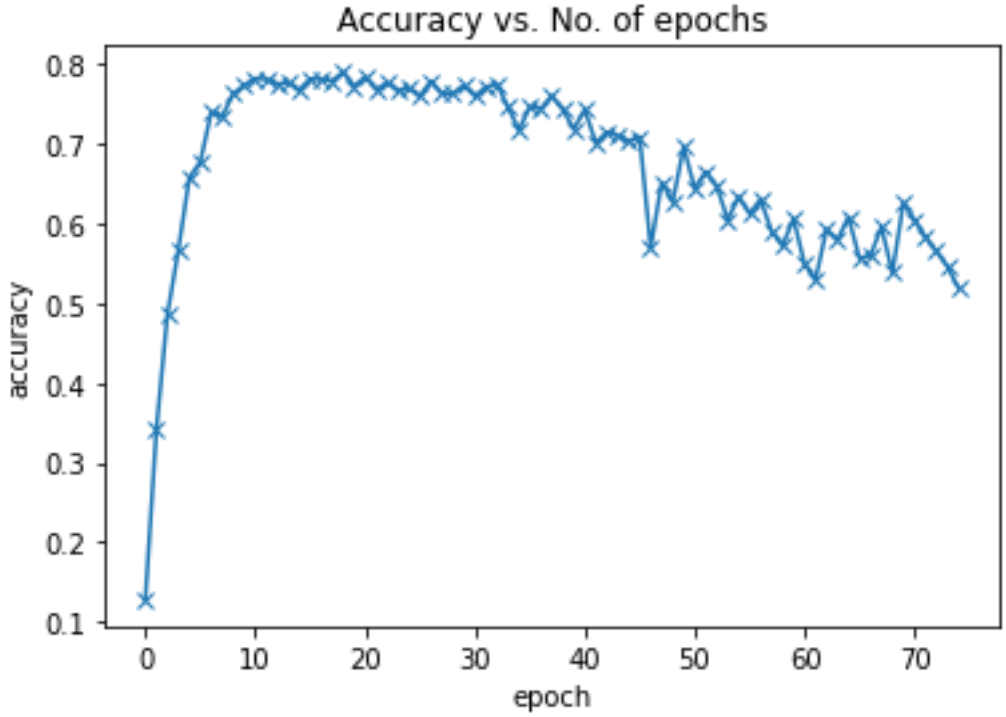
11) Model = 3 Convolutional layers, 4 Linear layers (MODEL3)

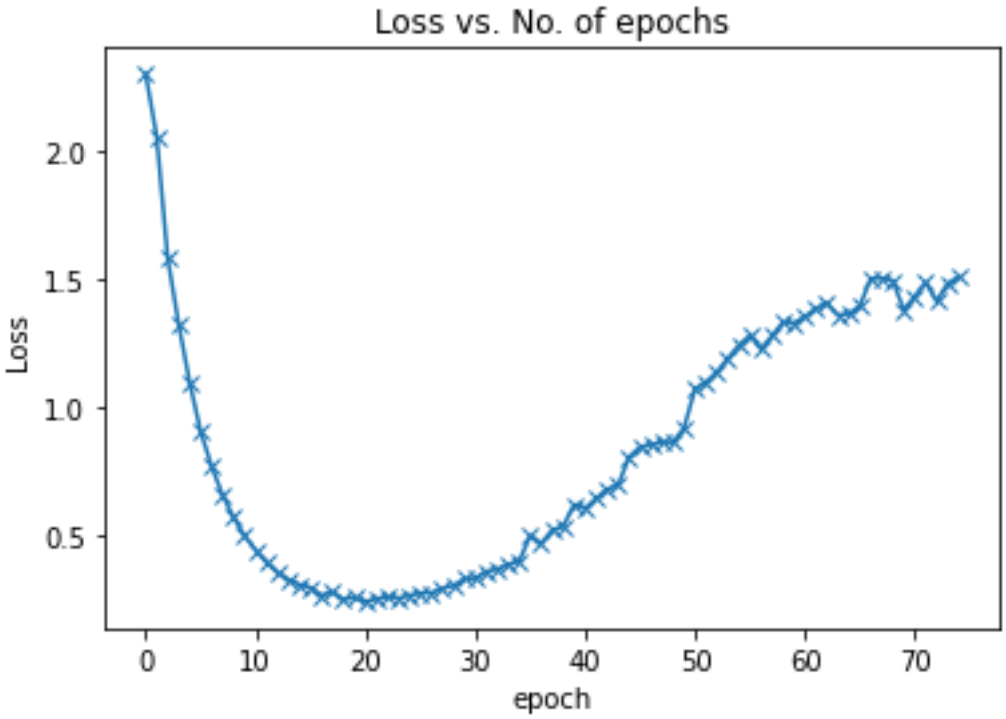
```
num_epochs = 75
optimizer = torch.optim.SGD
```
Learning rate = 0.05
momentum =0.9
activation = ReLU

exponential learning rate decay = 0.9 (gamma)

```
Epoch [17], train_loss: 0.2742, val_loss: 0.8649, val_acc: 0.7798
Epoch [18], train_loss: 0.2459, val_loss: 0.8374, val_acc: 0.7907
```

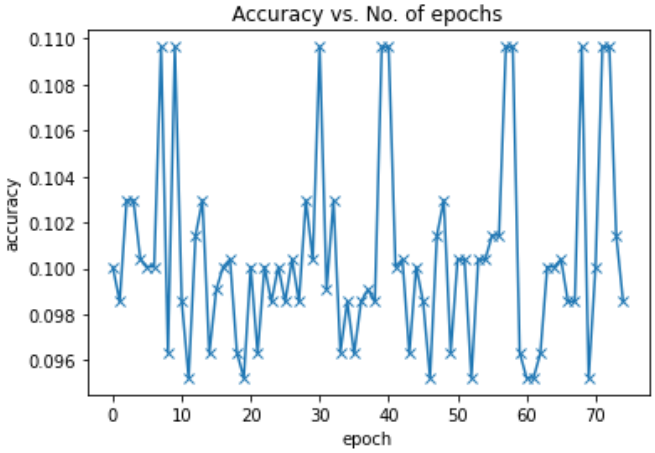
NOTE: The model converges and performs best at around 18 epochs



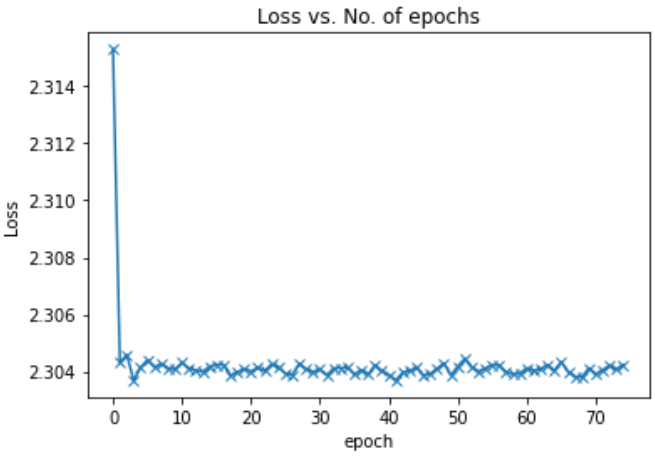


12) Model = 3 Convolutional layers, 4 Linear layers (MODEL3)

```
num_epochs = 75
optimizer = torch.optim.SGD
learning rate = 0.08
momentum = 0.9
Sigmoid activation
```




```
plot_loss(history)
```



13) Model = 3 Convolutional layers, 4 Linear layers (MODEL3)

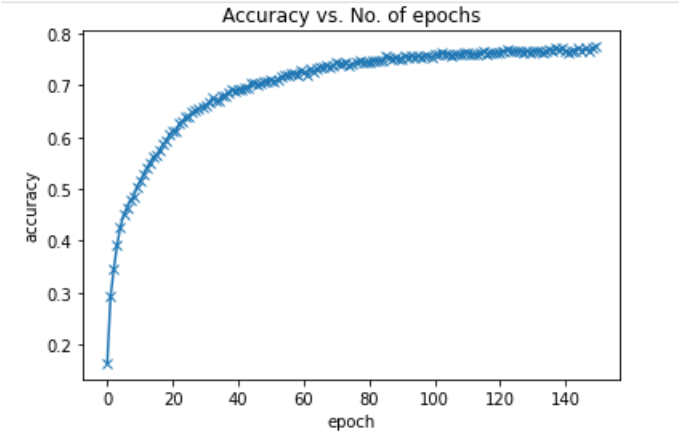
num_epochs = 150

optimizer = torch.optim.SGD

Learning rate = 0.001

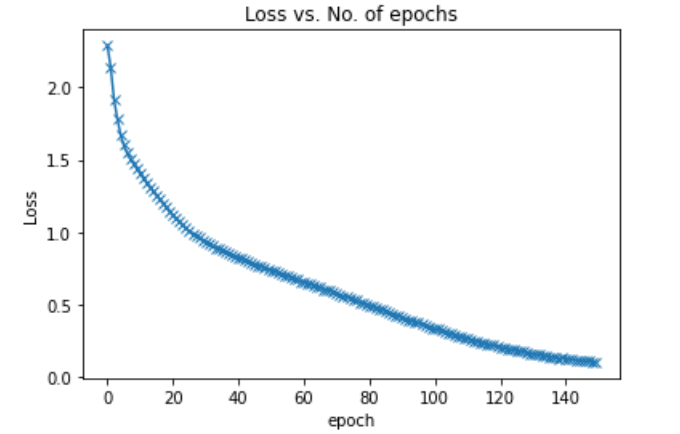
momentum = 0.9

ReLU activation



+ Code + Markdown

```
plot_loss(history)
```



Conclusion : There are many variables and parameters that can be used to tweak the given problem and get a better solution. Those being:

- Model used (countless possibilities for each layer)
- Choice of optimizer
- Number of epochs
- Batch size
- Tweaking the parameters of the optimizer

Best results were obtained with Adam optimizer, LR=0.001, Model1 with dropout probability of 40%

PART B

We picked a dataset containing images of spiders of 15 different categories.
https://www.kaggle.com/datasets/gpiosenka/yikes-spiders-15-species

We used the pretrained alexnet available to use from pytorch.models. We used the pretrained alexnet and set it to eval mode before using it on our images.

We extracted the features vector from the last layer/output layer of alexnet. We did so for the train and test set of images. Using these features we trained different classification models : LogisticRegression, SVM and Gaussian NB.

The accuracies obtained on the Spiders dataset were :

Serial No.	Classification Model	Accuracy Obtained
1	LogisticRegression	0.92
2	SVM	0.8533333333333334
3	Gaussian NB	0.7466666666666667

All the models give an accuracy of 1.0 on the BikeVsHorse dataset.

For the Spiders dataset, Logistic regression gives the best score followed by SVM, which is followed by GNB.

PART C: YOLO

YOLO (You Only Look Once) is a cutting-edge object detection technique that works in real time. That is, an algorithm for detecting and recognizing different things in a picture. Object detection in YOLO is done as a regression problem, and the identified class probabilities in pictures.

Convolutional neural networks (CNN) are used in the YOLO method to recognize objects in real time. To detect objects, the technique simply requires a single forward propagation through a neural network. This indicates that a single algorithm run is used to forecast the entire image. CNN is used to forecast multiple bounding boxes and class probabilities at the same time.

The YOLO algorithm consists of various variants. We used YOLOv5 for this assignment's third part. It is the latest version of YOLO available.

YOLO algorithm uses three techniques:

1. Residual blocks — The image is first separated into several grids. The dimensions of each grid are S × S.
2. Bounding box regression — A bounding box is an outline that draws attention to a certain object in a picture. The following attributes are present in every bounding box in the image:
    a. Width
    b. Height
    c. Class
    d. Bounding box center
3. Intersection Over Union (IOU) — The concept of intersection over union (IOU) illustrates how boxes overlap in object detection. IOU is used by YOLO to create an output box that properly surrounds the element. The bounding boxes and their confidence scores are predicted by each grid cell. If the anticipated and real bounding boxes are identical, the IOU is 1. This approach removes bounding boxes that aren't the same size as the actual box. NMS (Non Maximum Suppression) is hence utilized for picking one bounding box out of a slew of overlapping ones. If objects are in the bounding box of another object , we use concepts such as anchor boxes.

Where YOLO works well:

- YOLO is very fast as compared to other methods of object detection like sliding window object detection, RCNN, Fast RCNN, Faster RCNN. Process frames at a rate of 45 to 150 frames per second (depending on the size of the network), which is faster than real-time.
- The network is able to generalise the image better.

Where YOLO Fails:

- When compared to Faster RCNN, it has a lower recall and a higher localization error.
- Because each grid can only propose two bounding boxes hence it has difficulty detecting nearby items.
- Small items are difficult to notice.

Hyperparameters and Observation:

```
!python train.py --img 416 --batch 16 --epochs 150 --data {dataset.location}/data.yaml
--weights yolov5s.pt --cache
```

Here, we are able to pass a number of arguments:

- Image size: 416
- Batch Size: 16
- The number of Training Epochs: 150
- Weights: *YOLOv5s of* YOLOv5 (we need to have a trade-off between the speed and accuracy).
- cache: cache images for faster training

```
!python detect.py --weights runs/train/exp/weights/best.pt --img 416 --conf 0.7
--source {dataset.location}/test/images
```

Confidence_threshold = 0.7

There was a trade-off between the time taken and accuracy. The more epochs we take, the better the results.

Some Outputs

Conclusion

YOLO is an excellent technique for object detection, but fine-tuning the hyperparameters may take a lot of time. The confidence factor is a threshold for better accuracy which should be high enough to avoid false positives. But this may lead to false negatives.

Reference

https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/

https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/

https://www.kaggle.com/shadabhussain/cifar-10-cnn-using-pytorch

https://pytorch.org/docs/stable/index.html

https://www.youtube.com/watch?v=ag3DLKsl2vk&t=1s

https://www.geeksforgeeks.org/yolo-you-only-look-once-real-time-object-detection/

https://colab.research.google.com/github/roboflow-ai/yolov5-custom-training-tutorial/blob/main/yolov5-custom-training.ipynb#scrollTo=jtmS7_TXFsT3

https://www.kaggle.com/code/charel/yolov5-1st-place-world-championships-robocup-2021/notebook

Team Members

Abhinav H Kamath (IMT2019001)
Archit Sangal(IMT2019012)
Phani SriRam(IMT2019514)