Image Classification on CIFAR-100

Assignment 4

Devansh Gupta **20171100**

Problem Statement

Image classification is to classify different images into different classes. It doesn't matter here to put them in the selection box. Points to consider:

```
class Model(nn.Module):
  def __init__(self):
    super(Model,self).__init__()
    self.conv1 = nn.Sequential(nn.Conv2d(3,32,3,padding=1),
                               nn.ReLU(inplace=True),
                               nn.MaxPool2d(2,2))
    self.conv2 = nn.Sequential(nn.Conv2d(32,64,3,padding=1),
                               nn.ReLU(inplace=True),
                               nn.MaxPool2d(2,2))
    self.conv3 = nn.Sequential(nn.Conv2d(64,128,3,padding=1),
                               nn.ReLU(inplace=True),
                               nn.MaxPool2d(2,2))
    self.fc1 = nn.Sequential(nn.Linear(2048,1024),
                             nn.ReLU(inplace=True))
    self.fc2 = nn.Sequential(nn.Linear(1024,512),
                             nn.ReLU(inplace=True))
    self.fc3 = nn.Linear(512,100)

  def forward(self,x):
    # print(x.shape)
    x = self.conv1(x)
    # print(x.shape)
    x = self.conv2(x)
    # print(x.shape)
    x = self.conv3(x)
    # print(x.shape)
    fc = x.view(x.size(0),-1)
    # print(fc.shape)
    fc = self.fc1(fc)
    fc = self.fc2(fc)
    out = self.fc3(fc)
    return out
```

Use Pytorch as an encoding framework. All experiments are performed on Google Collab for over 50 epochs . Unless otherwise specified, i have used Adam Optimizer with LR (1e-3) with cross entropy loss. The image is normalized with

mean=[0.507, 0.487, 0.441], std=[0.267, 0.256, 0.276]

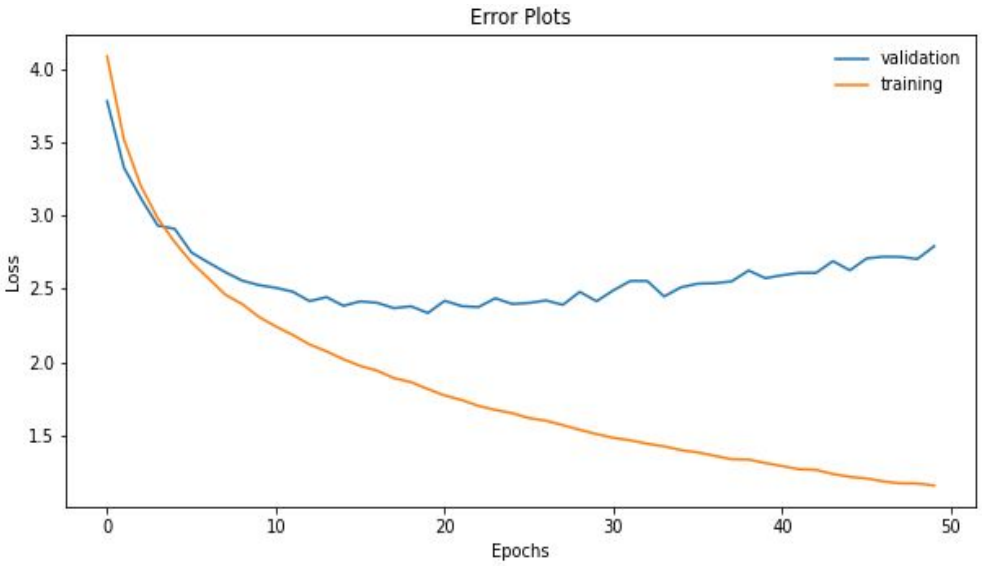
Training validation split is 80-20%.

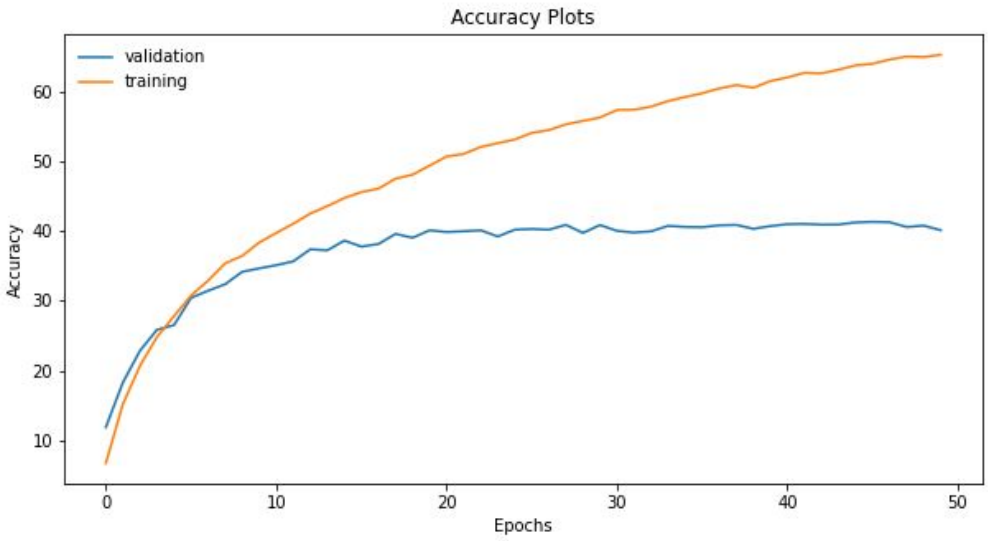
Base Architecture

The network consists of 3 convolution layers after which there is ReLU activation and Maxpool. The learnable parameters are only in Convolution Layers and Fully Connected Layers.

Results

As it can be seen, the validation stops after a certain point and the training loss keeps on decreasing. I have perform early stopping to ensure we have the model with lowest validation error.





Training loss goes as high as to 65% whereas validation loss remains in the region of 40-42%.

Test Accuracy on the network = **43.68%**

Base Architecture with Batch Norm

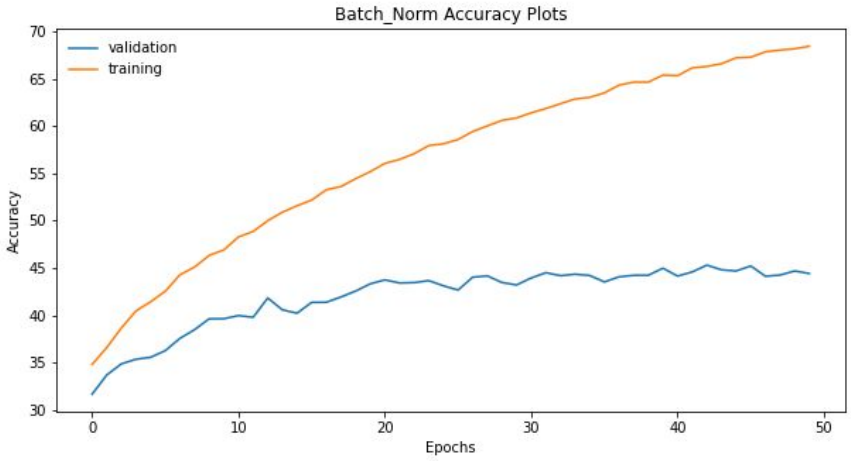
We now add batch norm layers after all the convolution layers. The idea is that if we normalize the inputs in order to improve training why cant we do that while giving input to the hidden layers. It stabilizes training and acts as a sort of a regularizer.

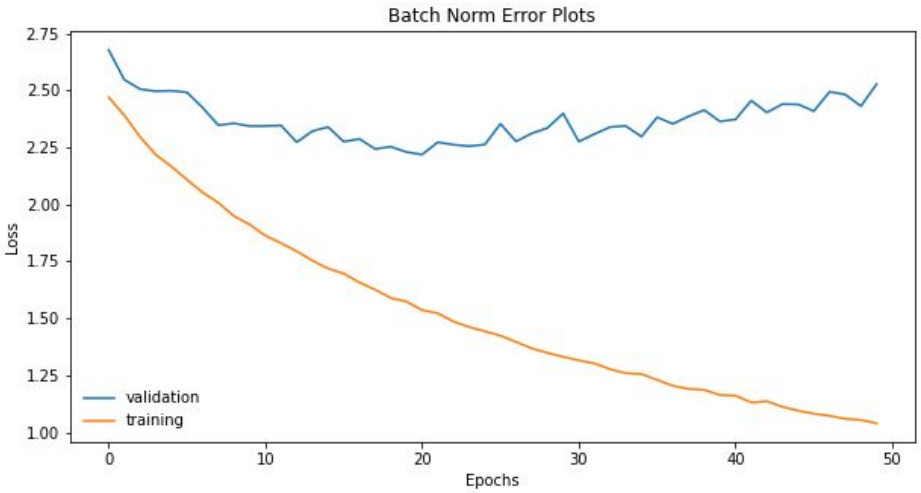
Batch normalization reduces the amount by what the hidden unit values shift around.

```
class Model(nn.Module):
  def __init__(self):
    super(Model,self).__init__()
    self.conv1 = nn.Sequential(nn.Conv2d(3,32,3,padding=1),
                               nn.BatchNorm2d(32),
                                nn.ReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv2 = nn.Sequential(nn.Conv2d(32,64,3,padding=1),
                               nn.BatchNorm2d(64),
                                nn.ReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv3 = nn.Sequential(nn.Conv2d(64,128,3,padding=1),
                               nn.BatchNorm2d(128),
                                nn.ReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.fc1 = nn.Sequential(nn.Linear(2048,1024),
                               nn.ReLU(inplace=True))
    self.fc2 = nn.Sequential(nn.Linear(1024,512),
                               nn.ReLU(inplace=True))
    self.fc3 = nn.Linear(512,100)

  def forward(self,x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    fc = x.view(x.size(0),-1)
    fc = self.fc1(fc)
    fc = self.fc2(fc)
    out = self.fc3(fc)
    return out
```

Results

The test accuracy we get is **46.76%**.



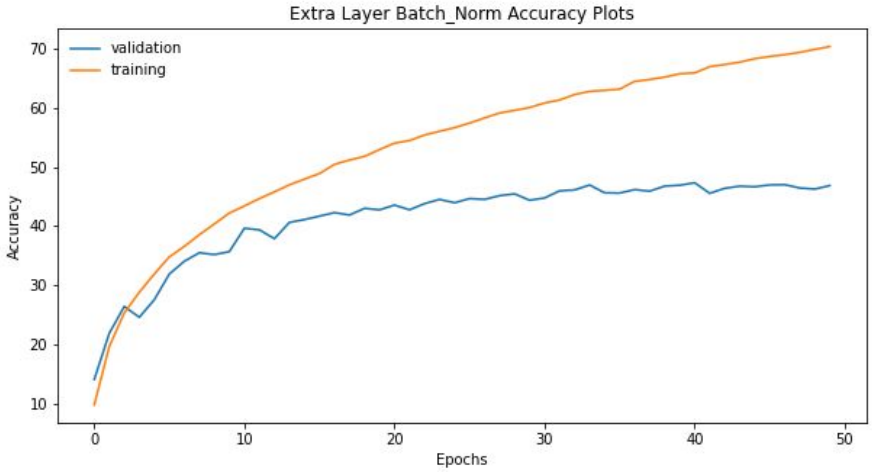


Base + Conv+ BatchNorm (New Base)

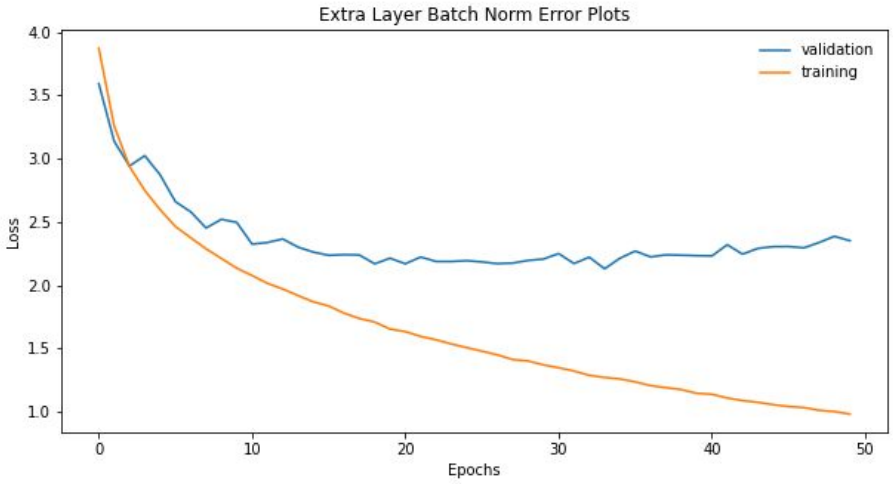
We add another convolution layer to compare our performances. So far it gives the best performance. However deeper networks are not necessarily better which will be shown in the next experiment. We will also use this as our **New Base**.

Results

The test accuracy we get is **49.50%.**



```
class Model(nn.Module):
  def __init__(self):
    super(Model,self).__init__()
    self.conv1 = nn.Sequential(nn.Conv2d(3,32,3,padding=1),
                               nn.BatchNorm2d(32),
                                nn.ReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv2 = nn.Sequential(nn.Conv2d(32,64,3,padding=1),
                               nn.BatchNorm2d(64),
                                nn.ReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv3 = nn.Sequential(nn.Conv2d(64,128,3,padding=1),
                               nn.BatchNorm2d(128),
                                nn.ReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv4 = nn.Sequential(nn.Conv2d(128,256,3,padding=1),
                               nn.BatchNorm2d(256),
                                nn.ReLU(inplace=True),
                                 )
    self.fc1 = nn.Sequential(nn.Linear(256*4*4,1024),
                               nn.ReLU(inplace=True))
    self.fc2 = nn.Sequential(nn.Linear(1024,256),
                               nn.ReLU(inplace=True))
    self.fc3 = nn.Linear(256,100)

  def forward(self,x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    x = self.conv4(x)
    # print(x.shape)
    fc = x.view(x.size(0),-1)
    # print(fc.shape)
    fc = self.fc1(fc)
    fc = self.fc2(fc)
    out = self.fc3(fc)
    return out
```



New Base and Linear and Dropout:

We add a new Linear Layer in the network (Fully Connected Layer) as well as Dropout.

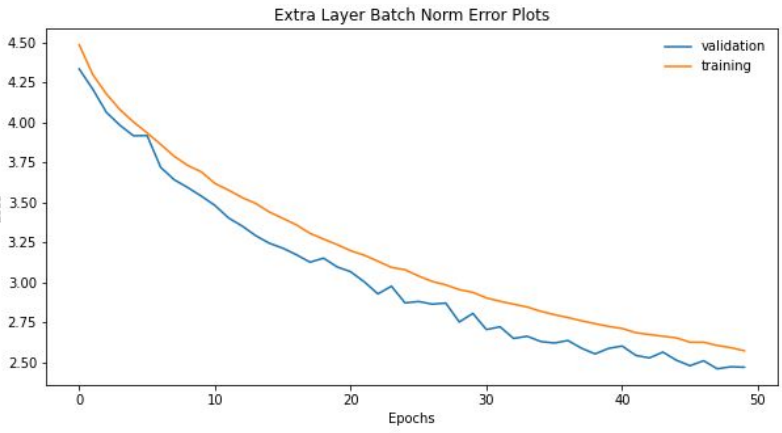
Dropout refers to ignoring neurons. During the training phase of certain set of neurons which is chosen at random.

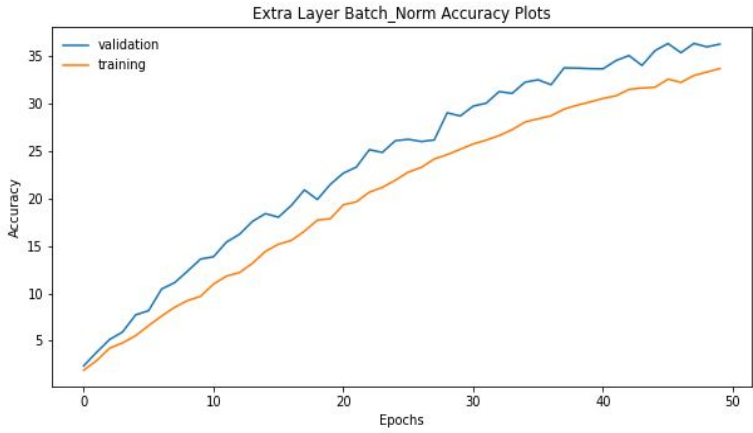
Dropout is useful to prevent overfitting. A fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data.

This network doesnt converge in 50 epochs and hence gives a poorer result as compared to others.

Results

The test accuracy we get is **38.40%.**





New Base with ELU:

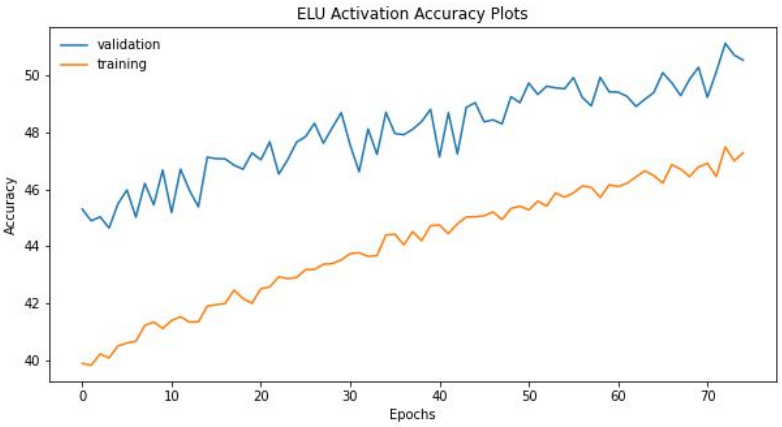
Exponential Linear Unit or its widely known name ELU is a function that tends to converge cost to zero faster and produce more accurate results. ELU is very similar to RELU except negative inputs. They are both in identity function form for non-negative inputs. On the other hand, ELU becomes smooth slowly until its output is equal to -α whereas RELU sharply smoothes.

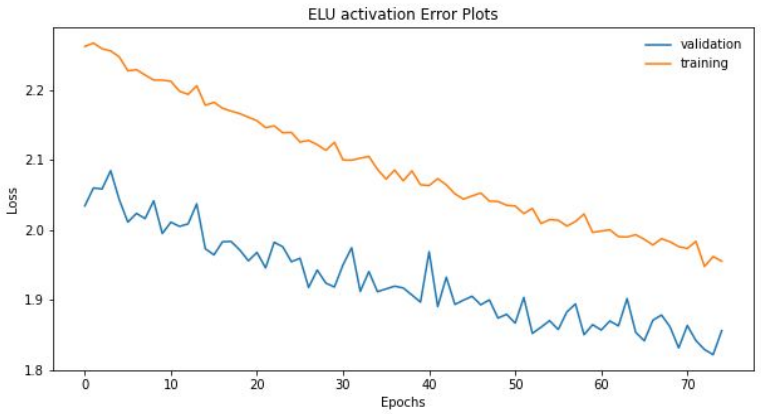
We run this for 75 epochs due to the weird behaviour that we observe,valid_loss<train_loss

```
class Model(nn.Module):
  def __init__(self):
    super(Model,self).__init__()
    self.conv1 = nn.Sequential(nn.Conv2d(3,32,3,padding=1),
                               nn.BatchNorm2d(32),
                                nn.ELU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv2 = nn.Sequential(nn.Conv2d(32,64,3,padding=1),
                               nn.BatchNorm2d(64),
                                nn.ELU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv3 = nn.Sequential(nn.Conv2d(64,128,3,padding=1),
                               nn.BatchNorm2d(128),
                                nn.ELU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv4 = nn.Sequential(nn.Conv2d(128,256,3,padding=1),
                               nn.BatchNorm2d(256),
                                nn.ELU(inplace=True),
                                 )

    self.fc1 = nn.Sequential(nn.Linear(256*4*4,1024),
                               nn.ELU(inplace=True),
                              nn.Dropout())
    self.fc2 = nn.Sequential(nn.Linear(1024,256),
                               nn.ELU(inplace=True),
                              nn.Dropout())
    self.fc3 = nn.Linear(256,100)

  def forward(self,x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    x = self.conv4(x)
    # print(x.shape)
    fc = x.view(x.size(0),-1)
    # print(fc.shape)
    fc = self.fc1(fc)
    fc = self.fc2(fc)
    out = self.fc3(fc)
    return out
```

Results

The test accuracy we get is **56.87%**



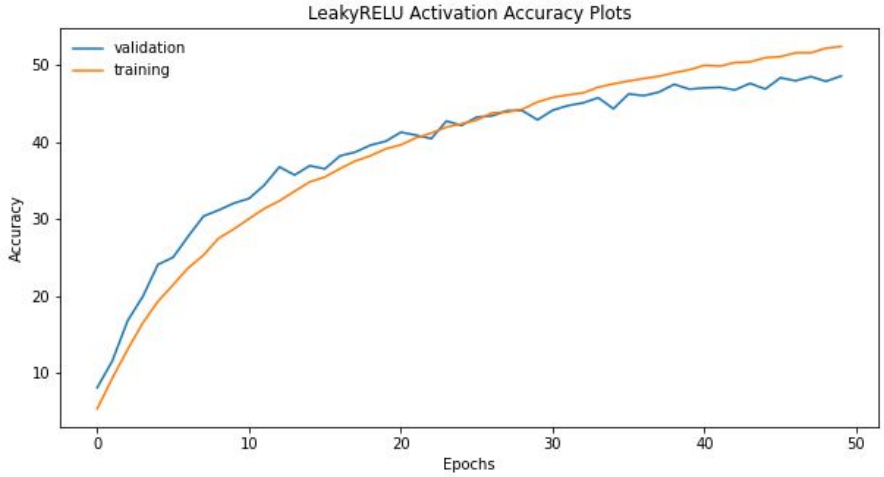


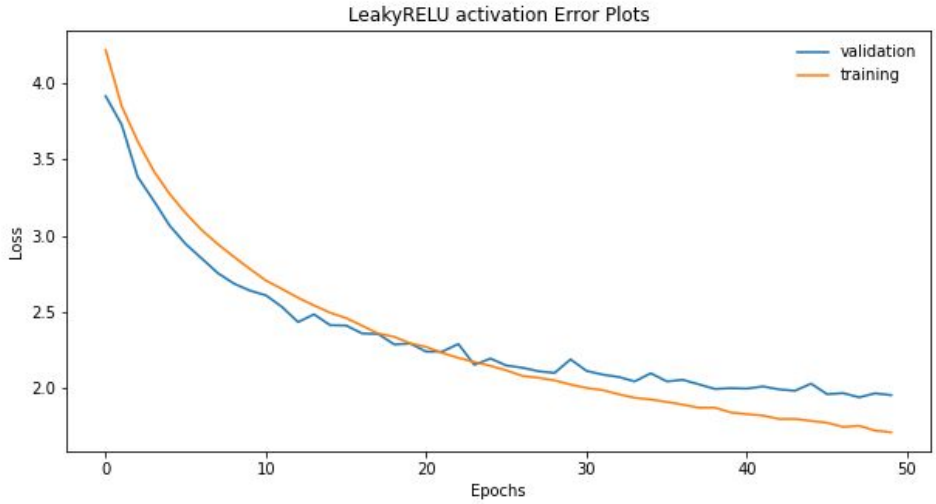
New Base with LeakyReLU

LeakyRelu is a variant of ReLU. Instead of being 0 when z<0 , a leaky ReLU allows a small, non-zero, constant gradient α . Leaky ReLUs are one attempt to fix the “dying ReLU” problem by having a small negative slope (of 0.01, or so).

Results

The test accuracy we get is **51.23%**




LeakyRELU activation Error Plots
validation
training
Loss
Epochs
4.0
3.5
3.0
2.5
2.0
0
10
20
30
40
50

```
class Model(nn.Module):
  def __init__(self):
    super(Model,self).__init__()
    self.conv1 = nn.Sequential(nn.Conv2d(3,32,3,padding=1),
                               nn.BatchNorm2d(32),
                                nn.LeakyReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv2 = nn.Sequential(nn.Conv2d(32,64,3,padding=1),
                               nn.BatchNorm2d(64),
                                nn.LeakyReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv3 = nn.Sequential(nn.Conv2d(64,128,3,padding=1),
                               nn.BatchNorm2d(128),
                                nn.LeakyReLU(inplace=True),
                                 nn.MaxPool2d(2,2))
    self.conv4 = nn.Sequential(nn.Conv2d(128,256,3,padding=1),
                               nn.BatchNorm2d(256),
                                nn.LeakyReLU(inplace=True),
                                 )

    self.fc1 = nn.Sequential(nn.Linear(256*4*4,1024),
                                nn.LeakyReLU(inplace=True),
                               nn.Dropout())
    self.fc2 = nn.Sequential(nn.Linear(1024,256),
                                nn.LeakyReLU(inplace=True),
                               nn.Dropout())
    self.fc3 = nn.Linear(256,100)

  def forward(self,x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = self.conv3(x)
    x = self.conv4(x)
    # print(x.shape)
    fc = x.view(x.size(0),-1)
    # print(fc.shape)
    fc = self.fc1(fc)
    fc = self.fc2(fc)
    out = self.fc3(fc)
    return out
```