STATISTICAL MACHINE LEARNING

PROJECT-REPORT

SHREYA SINHA-ss6415

PART 4: DEEP LEARNING

QUESTION 3:

Here I used Artificial Neural Network to train the model. I split the train data into train(50000 rows) and validation(10000 rows) with Batch Size 32

Model Name= model

Model architecture:

```
# Building Our Mode
class Network(nn.Module):
    # Declaring the Architecture
    def __init__(self):
        super(Network,self).__init__()
        self.fc1 = nn.Linear(28*28, 100)
        self.fc2 = nn.Linear(100, 10)

    # Forward Pass
    def forward(self, x):
        x = x.view(x.shape[0],-1)    # Flatten the images
        x = F.relu(self.fc1(x))
        #x = F.relu(self.fc2(x))
        x = self.fc2(x)
        return x

model = Network()
```

Number of layers- 2 linear layer and 1 Relu layer. The first linear layer is of the size (28*28, 100) and second linear layer is of the size (100,10)

Part a and b:

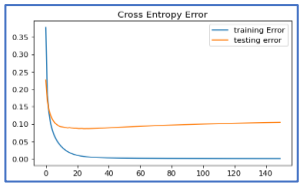
1. Random Seed 1

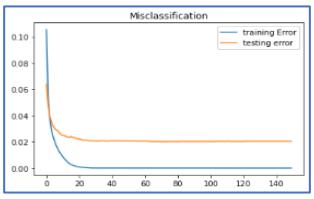
Training Cross Entropy Loss: 0.0003485154494326385

Validation Cross Entropy Loss: 0.10436115714963064

Model train Cross Entopy error= 0.0

Model test misclassfication= 0.02029999





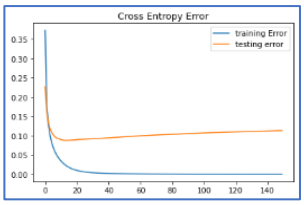
2. Random Seed 66

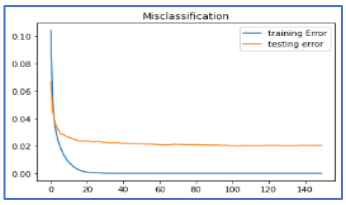
Training Loss: 0.0003443421145347052

Validation Loss: 0.113402044725914

Model train Misclassification = 0.0

Model test misclassification= 0.020299999999999985





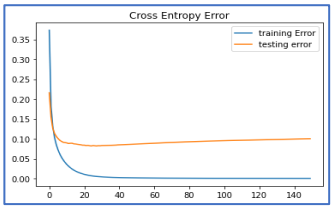
Random Seed 88

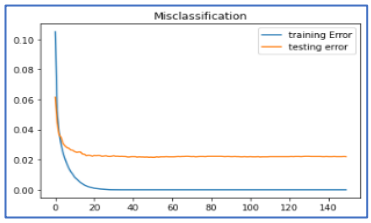
Training Cross Entropy Loss: 0.00034893627500318867

Validation Cross Entropy Loss: 0.1002857174073913

Model train Misclassification = 0.0

Model test Misclassification = 0.02200000000000002





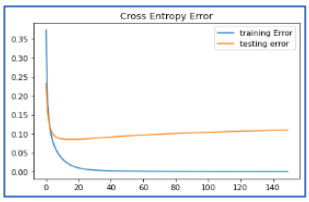
Random Seed 100

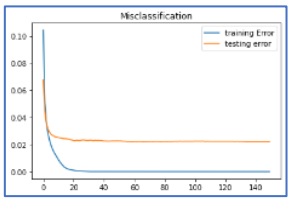
Training Cross Entropy Loss: 0.00034258958196807976

Validation Cross Entropy Loss: 0.10991273036555226

Model train Misclassification= 0.0

Model Test Misclassification = 0.02200000000000002





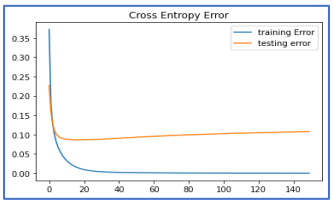
Random Seed 200

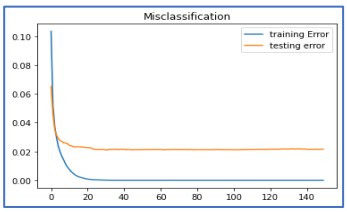
Training Cross Entropy Loss: 0.0003382573729833817

Validation Cross Entropy Loss: 0.10796993042024156

Model train Misclassification = 0.0

Model test Misclassification = 0.021599999999999953

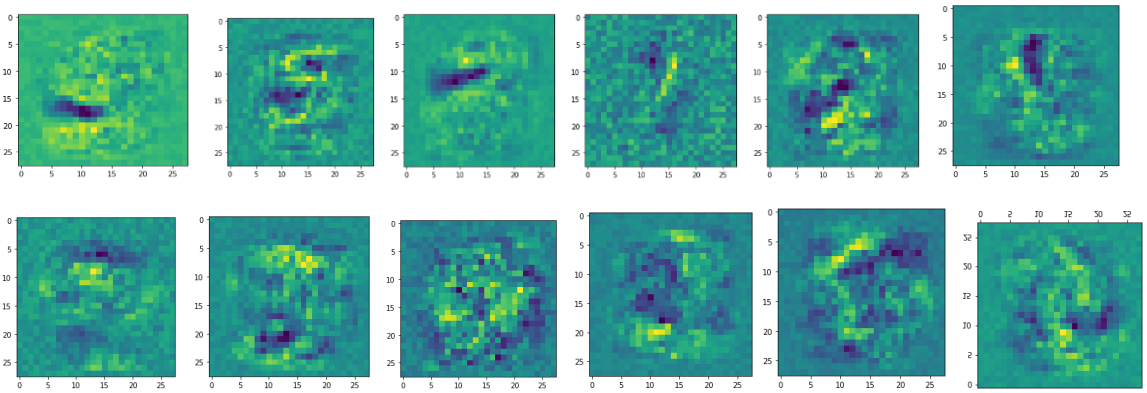


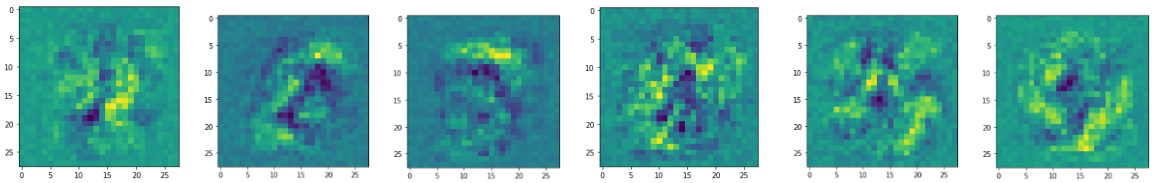


BEST MODEL OVER HERE IS AT SEED 66 with a misclassification error of 0.020299999999999985. This shows there is 98% accuracy

Part c

We visualized the results of the first linear layer of model at seed 66. It actually consists of 100 images. Below are some examples of the images. We can see a certain type of structure in some of them which depicts a number. For Example in first row figure 2 we can see the number 5 being shown. However some of the images are still unclear of what the numbers really are.

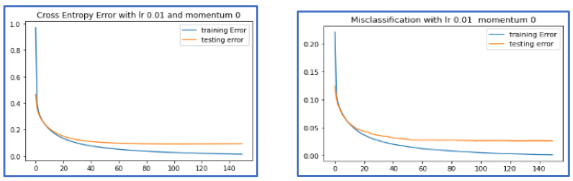




Part d

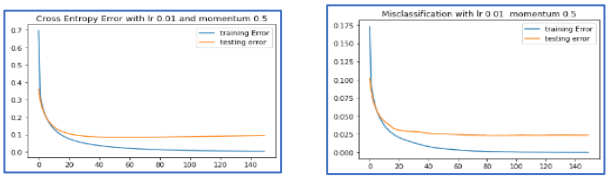
1. LR=0.01 and Momentum 0:

Training Loss: 0.014283372968414731
Validation Loss: 0.09362942037732748
Misclassification train error= 0.0013999999999999568
Misclassification test error = 0.02639999999999998



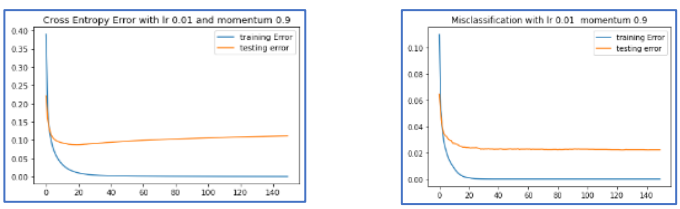
2. LR=0.01 and Momentum 0.5

Training Loss: 0.004146667242957585
Validation Loss: 0.09427163254315002
Misclassification train error = 4.0000000000040004e-05
Misclassification test error = 0.023599999999999954



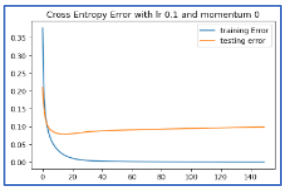
3. LR=0.01 and Momentum 0.9

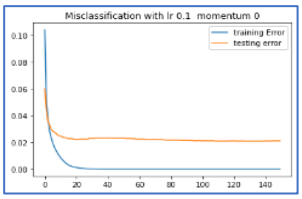
Training Loss: 0.00034411636448379926
Validation Loss: 0.11159407823284824
Misclassification train error= 0.0
Misclassification test error = 0.022299999999999986



4. LR=0.1 and Momentum 0:

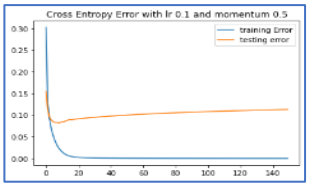
Training Loss: 0.0003462366317056811
Validation Loss: 0.09840100134322612
Misclassification train error = 0.0
Misclassification test error = 0.02100000000000002

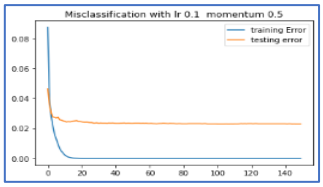




5. LR=0.1 and Momentum 0.5

Training Loss: 0.00013607018957934673
Validation Loss: 0.11302744622585974
Misclassification train error = 0.0
Misclassification test error = 0.02290000000000003

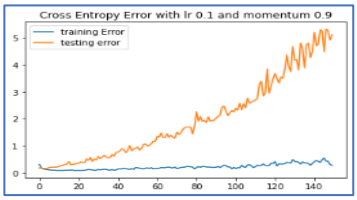


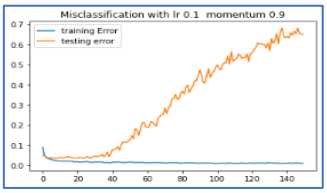


6. LR=0.1 and Momentum 0.9

Training Loss: 0.27000282672672715
Validation Loss: 5.124067403533029

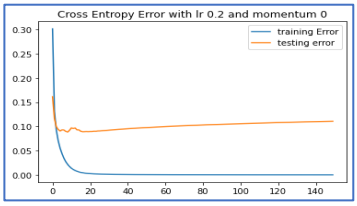
Model train Accuracy = 0.009099999999999997
Misclassification test error = 0.6496999999999999

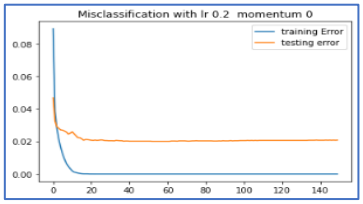




7. LR=0.2 and Momentum 0:

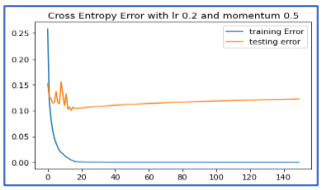
Training Loss: 0.00013976644261927824
Validation Loss: 0.11068924103044785
Model train Accuracy = 0.0
Model valid Accuracy = 0.02080000000000004

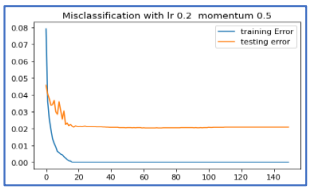




8. LR=0.2 and Momentum 0.5

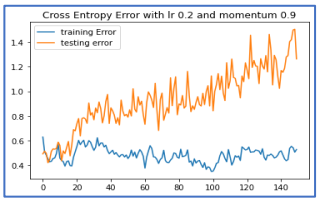
Training Loss: 4.9315120817183526e-05
Validation Loss: 0.12238614480807723
Model train Accuracy = 0.0
Model valid Accuracy = 0.02080000000000004

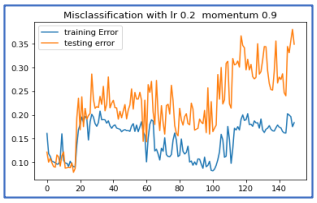




9. LR=0.2 and Momentum 0.9

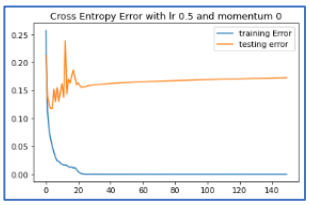
Training Loss: 0.5284352608821137
Validation Loss: 1.2633902159647439
Model train Accuracy = 0.18381999999999998
Model valid Accuracy = 0.34830000000000005

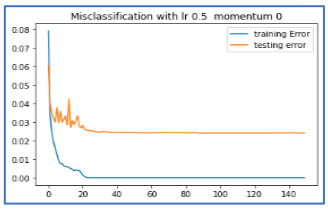




10. LR=0.5 and Momentum 0:

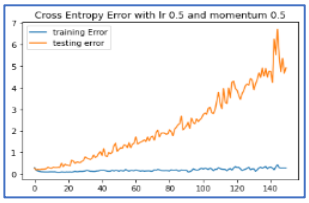
Training Loss: 2.1809093532665927e-05
Validation Loss: 0.1725947939884011
Model train Accuracy = 0.0
Model valid Accuracy = 0.02410000000000001

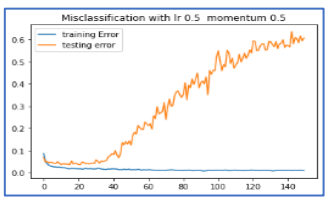




11. LR=0.5 and Momentum 0.5

Training Loss: 0.27161405819796763
Validation Loss: 4.92536942813458
Model train Accuracy = 0.00875999999999999
Model valid Accuracy = 0.6075999999999999





MODEL SELECTION :

We will hence select the Model with LR=0.2 and Momentum 0, It has a validation Cross entropy error of 0.11068924103044785 and misclassification error of 0.02080000000000004 . This was the model that gave the lowest test misclassification error

QUESTION 4 :

Here I used Deep Neural Network to train the model. I split the train data into train(50000 rows) and validation(10000 rows) with Batch Size 32

Model Name= model

Model architecture:

```
# Building Our Model
class Network(nn.Module):
    # Declaring the Architecture
    #Initialization
    def __init__(self):
        super(Network,self).__init__()
        self.conv1=nn.Conv2d(1,8,kernel_size=3,stride=1,padding=1)
        self.maxpool=nn.MaxPool2d(2, 2) # brackets taken out
        self.flat=nn.Flatten()
        self.fc1 = nn.Linear(8*14*14, 100)
        self.fc2 = nn.Linear(100, 10)
        self.relu=nn.ReLU()

    # Forward Pass
    def forward(self, x):

        x= self.relu(self.conv1(x))
        x=self.maxpool(x)
        x=self.flat(x)
        x=self.fc1(x)
        x=self.relu(x)
        x=self.fc2(x)
        return x
```

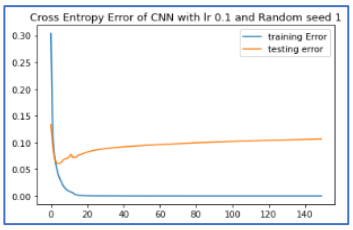
First layer is : Convolution layer. Secondly I applied Relu to it and then I applied maxpool on it. I then flattened the layer and applied linear layer(8*14*14,100) to it. Again followed by another relu. And again a linear layer(100,10).

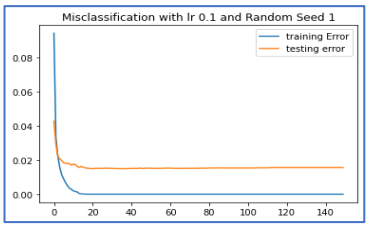
Number of layers are: 1 convolution Layer. 2 linear layers,1 output layer

PART A AND B

RANDOM SEED 1

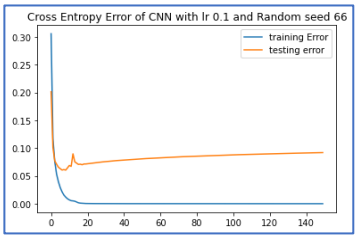
Training Loss: 1.995312300317693e-05
Validation Loss: 0.10651211529693136
Model train Misclassification = 0.0
Model test Misclassification = 0.015599999999999947

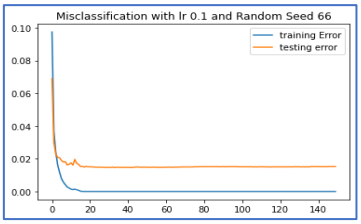




RANDOM SEED 66

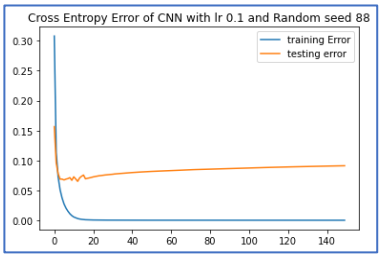
Training Loss: 1.8774394787106465e-05
Validation Loss: 0.0919277858561472
train Misclassification = 0.0
test Misclassification = 0.01529999999999998

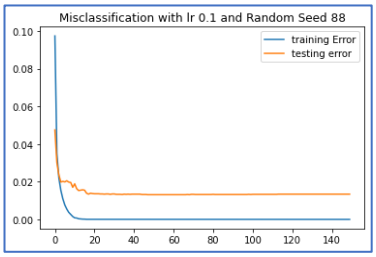




RANDOM SEED 88

Training Loss: 2.0207677073579574e-05
Validation Loss: 0.09115387806471703
train Misclassification = 0.0
test Misclassification = 0.013399999999999967

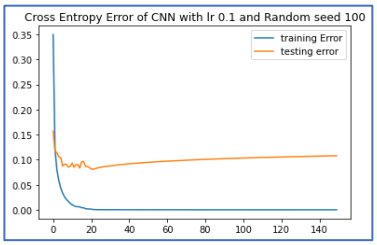


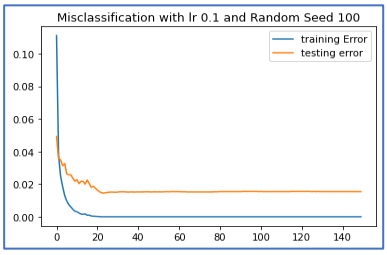


RANDOM SEED 100

Training Loss: 1.7887051364824673e-05

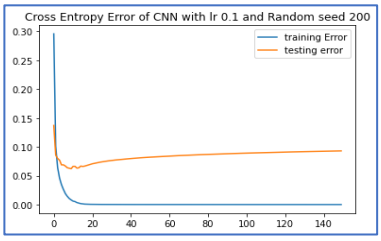
Validation Loss: 0.10793830492927185
train Misclassification = 0.0
test Misclassification = 0.01539999999999997

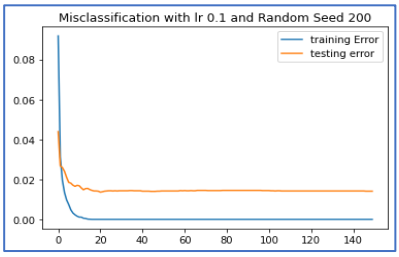




RANDOM SEED 200

Training Loss: 1.9974257278572282e-05
Validation Loss: 0.0929961737714039
train Misclassification = 0.0
test Misclassification = 0.014100000000000001





INTERPRETATION:

At random seed 88, this model gives the best answer with a test miclassification of test 0.013399999999999967. Hence we get an accuracy of 0.987 which is pretty high

How does network performuance differ in training vs validation?

Usually training has a lower accuracy than the validation .This is because our model is getting trained on the data. We can also see this by the plots.

Difference between cross entropy and Misclassification?

In some cases the cross entropy error starts to rise again whereas the misclassification error is still falling. This is because there is a difference in calculation. However in both cases train is lower than the testting error. Also misclassifcation is preffered to Cross entropy loss as its unbounded. And smal changes : For examle outlier can create havoc whereas that’s not the case in Misclassification eror.

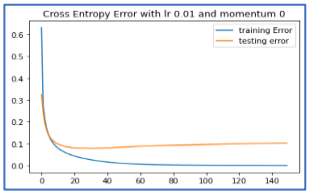
PART C

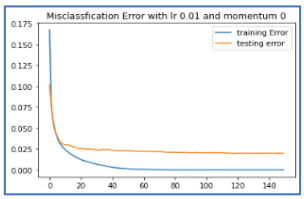
```
import matplotlib.pyplot as plt

params2 = list(model_new_cnn0.parameters())[0]
plt.figure(figsize=(8, 8))
for i in range(params2.shape[0]):
    plt.subplot(2,4,i + 1) # Since we know it is a 10 x 10 grid
    x = params2[i,:].detach().numpy()
    plt.imshow(x.reshape((3, 3)), cmap = "gray", interpolation = "nearest")
    plt.axis("off")
plt.subplots_adjust(wspace=0.25, hspace=0.01)
```

This shows visualization of weights in first layer. However we cant really see a structure per se except for placement of light , darkgrey etc

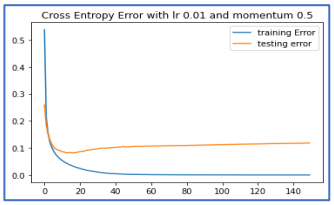
PART D:

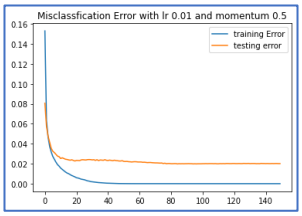
1. **LR=0.01 and Momentum 0:**
   Training Loss: 0.0006408033820785491
   Validation Loss: 0.10303375952616153
   Train Misclassification Error = 0.0
   Test Misclassification Error = 0.01970000000000005



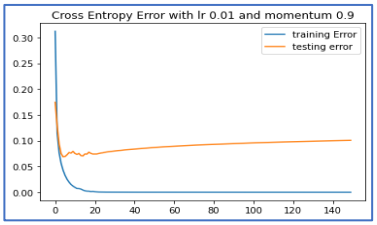


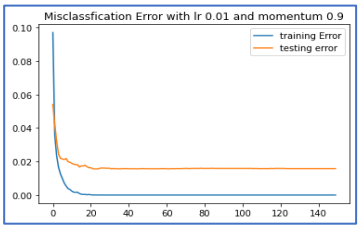
2. **LR=0.01 and Momentum 0.5:**
   Training Loss: 0.0001952669588016922
   Validation Loss: 0.11842244120121052
   Train Misclassification Error= 0.0
   Test Misclassification Error = 0.020100000000000007



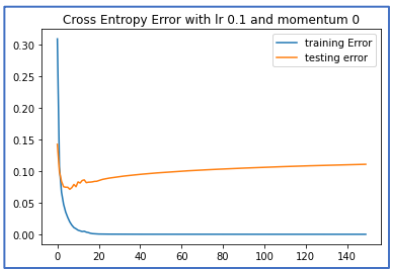


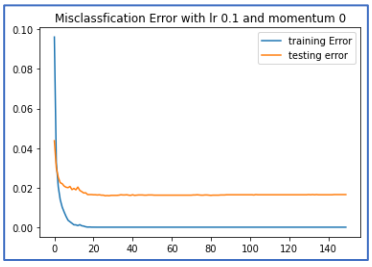
3. **LR=0.01 and Momentum 0.9:**
   Training Loss: 1.895077524790829e-05
   Validation Loss: 0.10057895236961238
   train Misclassification Error = 0.0
   Test Misclassification = 0.015800000000000036





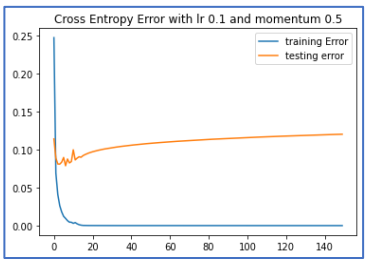
4. **LR=0.1 and Momentum 0:**
   Training Loss: 1.9009344880976435e-05
   Validation Loss: 0.11073513432879317
   Model train Accuracy = 0.0
   Model valid Accuracy = 0.01649999999999996

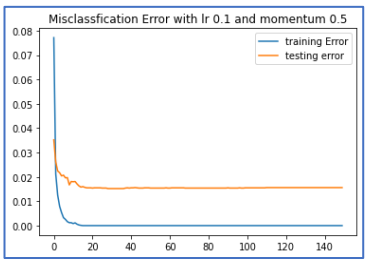




5. **LR=0.1 and Momentum 0.5:**
   Training Loss: 7.626290088760189e-06
   Validation Loss: 0.12035131436407052

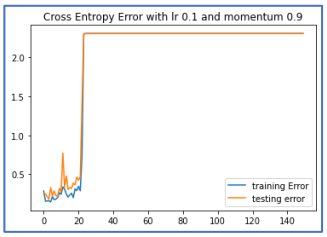
train Misclassification Error = 0.0
test Misclassification Error = 0.015599999999999947

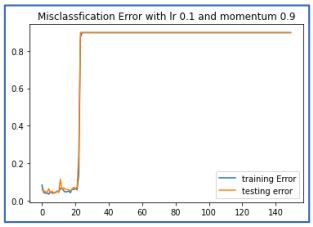




6. LR=0.1 and Momentum 0.9:

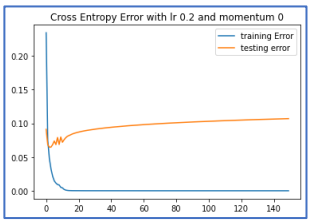
Training Loss: 2.3087022386906013
Validation Loss: 2.307242810916596
train Misclassification Error = 0.89686
test Misclassification Error = 0.8986

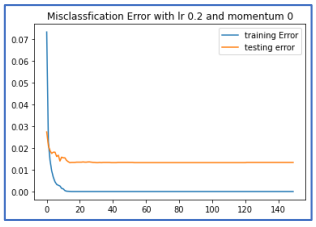




7. LR=0.2 and Momentum 0:

Training Loss: 8.234959201972216e-06
Validation Loss: 0.10695706154335075
Model train Accuracy = 0.0
Model valid Accuracy = 0.013399999999999967

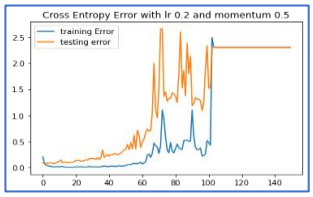


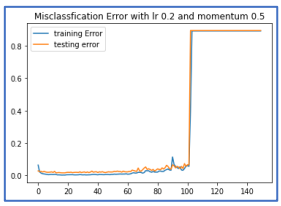


8. LR=0.2 and Momentum 0.5:

Training Loss: 2.3039336816210514
Validation Loss: 2.30694989198313

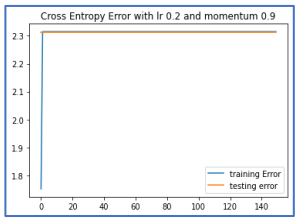
Model train Accuracy = 0.89264
Model valid Accuracy = 0.8951

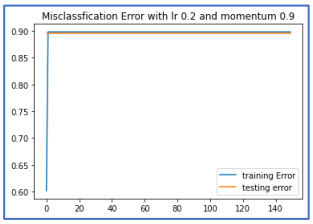




9. LR=0.2 and Momentum 0.9:

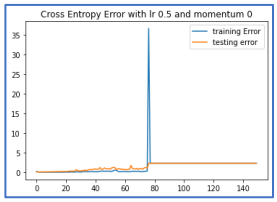
Training Loss: 2.3148364913974597
Validation Loss: 2.3130726623839846
Model train Accuracy = 0.89808
Model valid Accuracy = 0.8951

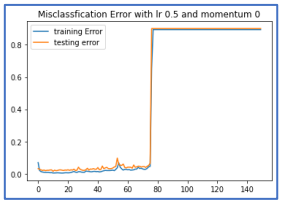




10. LR=0.5 and Momentum 0:

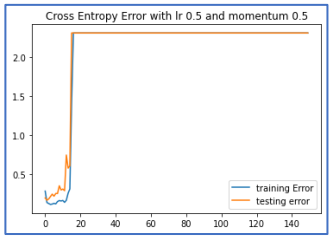
Training Loss: 2.304823670414725
Validation Loss: 2.305066560403988
Model train Accuracy = 0.8941
Model valid Accuracy = 0.9005

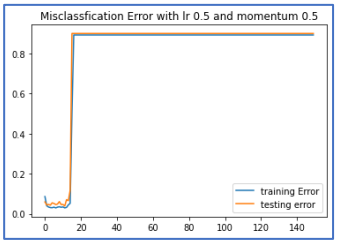




11. LR=0.5 and Momentum 0.5:

Training Loss: 2.3084053915430167
Validation Loss: 2.309103156050173
Model train Accuracy = 0.89284
Model valid Accuracy = 0.9005

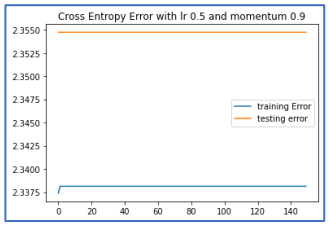


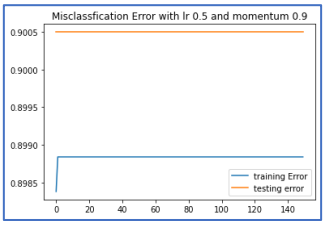


12. LR=0.5 and Momentum 0.9:

Training Loss: 2.3381343412612847

Validation Loss: 2.3547369542594154
Model train Accuracy = 0.89884
Model valid Accuracy = 0.9005





BEST SELECTED MODEL IS LR=0.01 and Momentum 0. It has a misclassification error of 0.01970000000000005 i.e 98.1% accuracy

Also gives a slightly better result than the single layer neural network . When we compare the best models CNN has 0.013399999999999967 error rate whereas single layer has 0.02. However there isnt a huge difference.

QUESTION 5

Favourite Deep learning:

Deep learning with 1 Convolution layer , 2 linear layer and Dropout is introduced.

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "*dropped out*." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "*view*" of the configured layer.

Here I used Deep Neural Network to train the model. I split the train data into train(50000 rows) and validation(10000 rows) with Batch Size 32

Model Name= model

Model architecture:

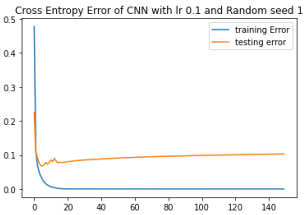
```python
# Building Our Mode
class Network(nn.Module):
    # Declaring the Architecture
    def __init__(self):
        super(Network,self).__init__()
        self.conv1=nn.Conv2d(1,8,kernel_size=3,stride=1,padding=1)
        self.maxpool=nn.MaxPool2d(2, 2) # brackets taken out
        self.flat=nn.Flatten()
        self.fc1 = nn.Linear(8*14*14, 100)
        self.fc2 = nn.Linear(100, 10)
        self.drop= nn.Dropout()
        self.relu=nn.ReLU()

    # Forward Pass
    def forward(self, x):

        x= self.relu(self.conv1(x))
        x=self.maxpool(x)
        x=self.flat(x)
        x=self.fc1(x)
        x=self.relu(x)
        x=self.drop(x)
        x=self.fc2(x)
        return x
```

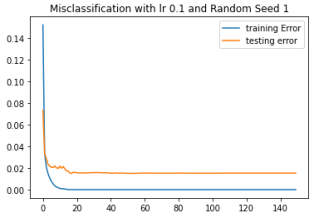
Number of layers are: 1 convolution Layer. 2 linear layers and 1 output. However we also are doing Max Pool ,1 flattened , 1 relu layer , and 1 dropout layer.

PART A AND B

RANDOM SEED 1

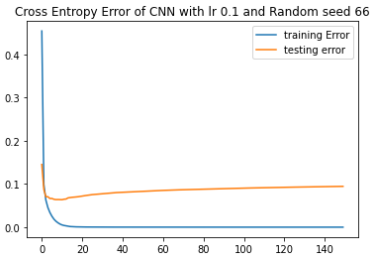
Training Loss: 2.0542979534472713e-05
Validation Loss: 0.10290628759776155
Model train Accuracy = 0.0
Model valid Accuracy = 0.015199999999999991

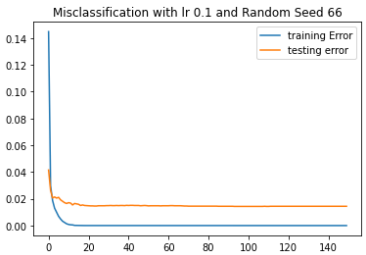




RANDOM SEED 66

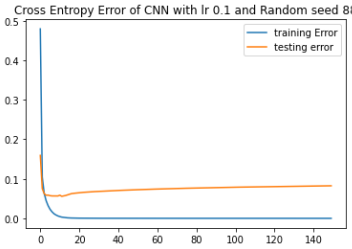
Training Loss: 2.2091797644587522e-05
Validation Loss: 0.09462894924454672
Model train Accuracy = 0.0
Model valid Accuracy = 0.014399999999999968

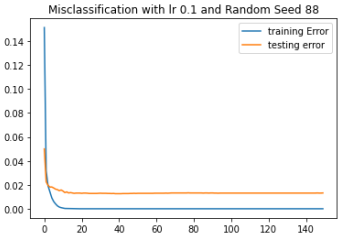




RANDOM SEED 88

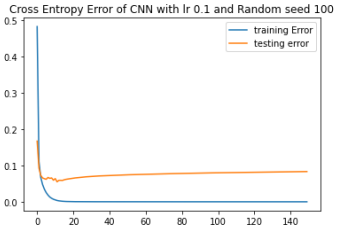
Training Loss: 2.2611858915447573e-05
Validation Loss: 0.08241212558082046
Model train Accuracy = 0.0
Model valid Accuracy = 0.01319999999999999

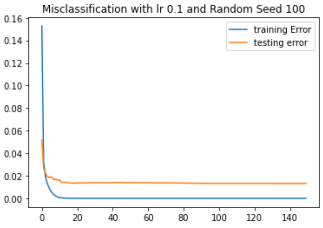




RANDOM SEED 100

Training Loss: 2.1808897050293505e-05
Validation Loss: 0.08327529260419275
Model train Accuracy = 0.0
Model valid Accuracy = 0.0131





RANDOM SEED 200

Training Loss: 2.798753625975291e-05
Validation Loss: 0.09444579014581422
Model train Accuracy = 0.0
Model valid Accuracy = 0.013900000000000023

RANDOM SEED 100 has the lowest misaccuracy weights

PART C

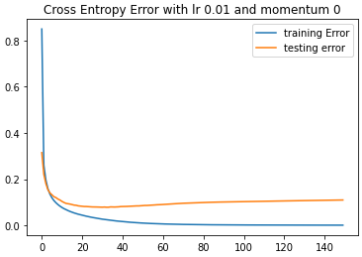
```
import matplotlib.pyplot as plt

params = list(model_new_cnn1.parameters())[0]
plt.figure(figsize=(8, 8))
for i in range(params.shape[0]):
    plt.subplot(2,4,i + 1) # Since we know it is a 10 x 10 grid
    x = params[i,:].detach().numpy()
    plt.imshow(x.reshape((3, 3)), cmap = "gray", interpolation = "nearest")
    plt.axis("off")
plt.subplots_adjust(wspace=0.25, hspace=0.01)
```

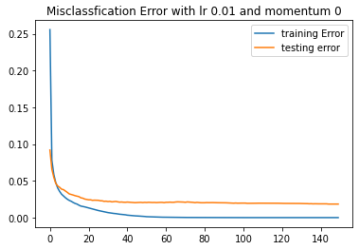
PART D:

1.LR=0.01 and Momentum 0

Training Loss: 0.0006326434913470427
Validation Loss: 0.10963230026176632
Model train Accuracy = 0.0

Model valid Accuracy = 0.018399999999999972

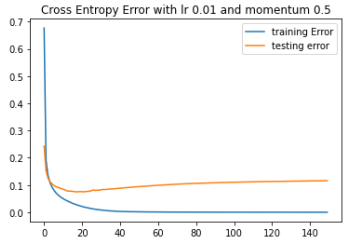


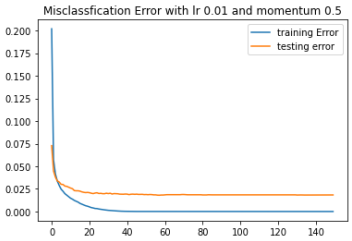


2.LR=0.01 and Momentum 0.5

Training Loss: 0.00017893673698882867
Validation Loss: 0.11596844277082073

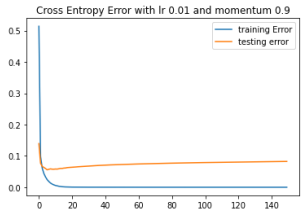
Model train Accuracy = 0.0
Model valid Accuracy = 0.018299999999999983

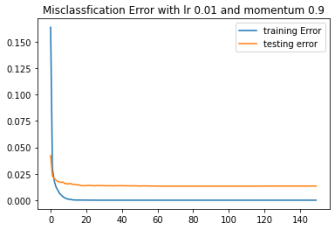




3. LR=0.01 and Momentum 0.9

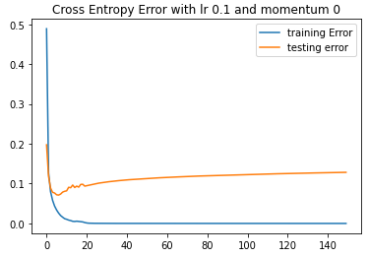
Training Loss: 2.2314315912551537e-05
Validation Loss: 0.08239334809357783
Model train Accuracy = 0.0
Model valid Accuracy = 0.013399999999999967

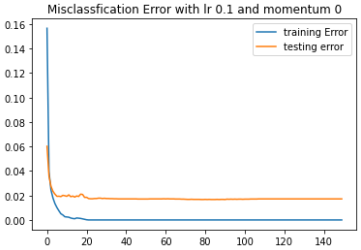




4. LR=0.1 and Momentum 0

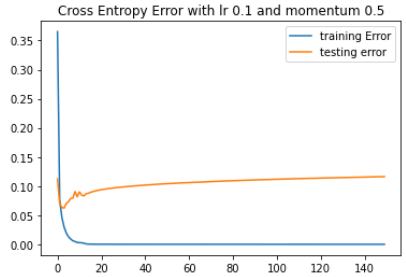
Training Loss: 1.928693232968974e-05
Validation Loss: 0.12833525710386395
Model train Accuracy = 0.0
Model valid Accuracy = 0.017199999999999993





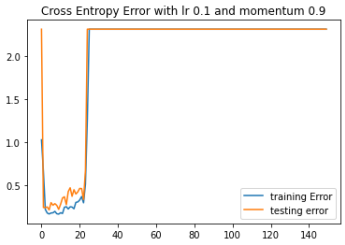
5. LR=0.1 and Momentum 0.5

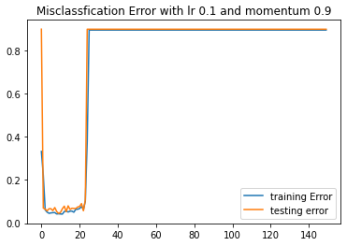
Training Loss: 9.039003450559894e-06
Validation Loss: 0.1162804716747233
Model train Accuracy = 0.0
Model valid Accuracy = 0.01539999999999997



6. LR=0.1 and Momentum 0.9

Training Loss: 2.308542891488347
Validation Loss: 2.308962955261572
Model train Accuracy = 0.89538
Model valid Accuracy = 0.8993





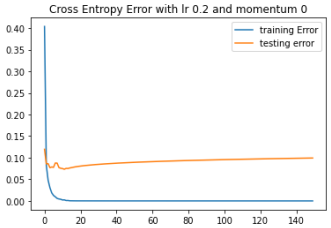
This shows that out model is overfitting and cant generalize error at all. However before 0.25 we can see training and test misclassification error. If we do early stopping at around 10 epoch we can see training and test error around only 0.05

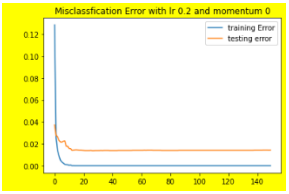
LR=0.2 and Momentum 0

Training Loss: 8.816294176595356e-06
Validation Loss: 0.09928045828987918

Model train Accuracy = 0.0

Model valid Accuracy = 0.01419999999999999

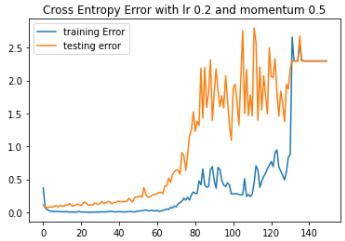


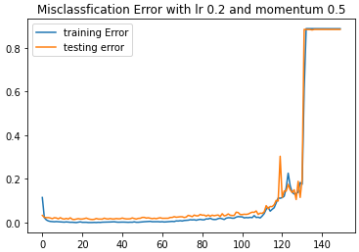


7. LR=0.2 and Momentum 0.5

Training Loss: 2.2965952684996758
Validation Loss: 2.2963461997790837
Model train Accuracy = 0.88854

Model valid Accuracy = 0.8852

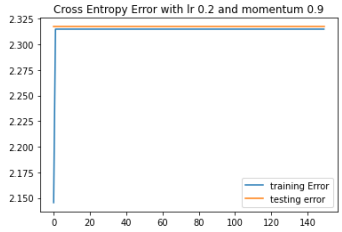


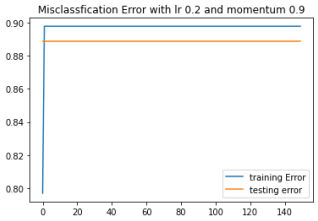


9.LR=0.2 and Momentum 0.9

Training Loss: 2.3149090028312522
Validation Loss: 2.3172620310189243

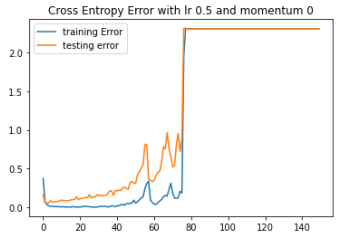
Model train Accuracy = 0.89768
Model valid Accuracy = 0.8886000000000001





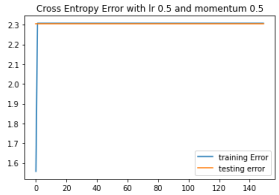
10. LR=0.5 and Momentum 0

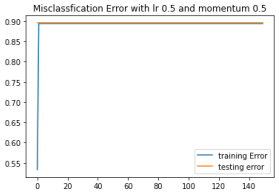
Training Loss: 2.304423138184648
Validation Loss: 2.303657174491273
Model train Accuracy = 0.89266
Model valid Accuracy = 0.8901



11. LR=0.5 and Momentum 0.5

Training Loss: 2.3079637245573603
Validation Loss: 2.3060715457501884
Model train Accuracy = 0.89416
Model valid Accuracy = 0.8956999999999999





HERE WE ARE PICKING LR=0.01 and Momentum 0.9 with the lowest Validation Loss: 0.08239334809357783 and Model Misaccuracy = 0.013399999999999967. That is 98.7% ACCURACY

Here we can see higher the momentum we can see more overfitting in some cases. This is because **Momentum** is an extension to the gradient descent optimization algorithm that allows the search to build inertia in a direction in the search space and overcome the oscillations of noisy gradients and coast across flat spots of the search space.

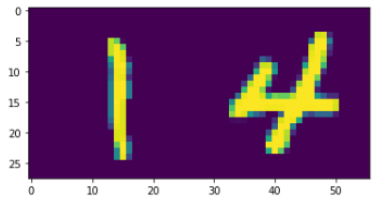
QUESTION 6

Here we imported the data .

Plotting of the Images:

```
] plt.imshow(X_train_7[0].reshape(28,56))
```

```
<matplotlib.image.AxesImage at 0x7f293ccebf10>
```

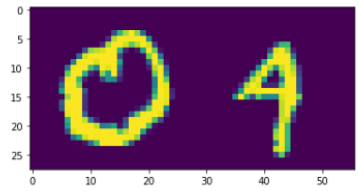


```
] y_train_7[0]
```

```
5
```

```
[17] plt.imshow(X_train_7[2].reshape(28,56))
```

```
<matplotlib.image.AxesImage at 0x7f4d52d1b690>
```



```
[18] y_train_7[2]
```

```
4
```

Interpretation:

- Firstly we see that the target function or y represent the sum of two numbers in the image
- The images are being scanned in a row wise way.

QUESTION 7

As this was a dataset and not a library like MNIST it didn't have predefined train and test loaders. Hence I first created a trainloader, testloader and validloader. I felt it was important as it helps in **parallelizing the data loading process with automatic batching** by using DataLoader. It is especially useful as I wanted to take batches as it helps in optimizing and coming to solutions faster.

MODEL 1

This Model includes Dropout,Maxpool,1 convolution layer and 2 linear layer

Here I used Deep Learning Neural Network to train the model. I split the train data into train(50000 rows) and validation(10000 rows) with Batch Size 32.I have introduced Dropout in this model.

Model Name= model

Model architecture:

```
# Building Our Model
class Network(nn.Module):
    # Declaring the Architecture
    def __init__(self):
        super(Network,self).__init__()
        self.conv1=nn.Conv2d(1,8,kernel_size=3,stride=1,padding=1)
        self.maxpool=nn.MaxPool2d(2, 2) # brackets taken out
        self.flat=nn.Flatten()
        self.fc1 = nn.Linear(3136, 100)
        self.fc2 = nn.Linear(100, 19)
        self.drop= nn.Dropout()
        self.relu=nn.ReLU()

    # Forward Pass
    def forward(self, x):

        x= self.relu(self.conv1(x))
        x=self.maxpool(x)
        x=self.flat(x)
        # print("Flat",x.shape)
        x=self.fc1(x)
        # print("FC1",x)
        x=self.relu(x)
        x=self.drop(x)
        x=self.fc2(x)
        return x
```

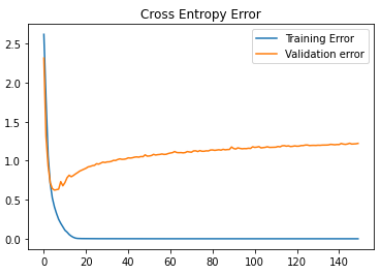
Number of layers:

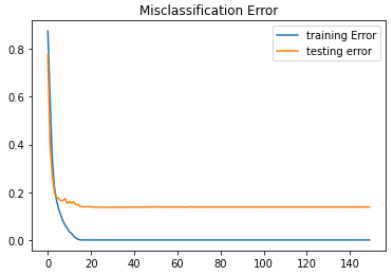
This Architecture contains 1 convolution layer. 1 maxpool,1 flattening,1 dropout and 2 linear layers of dimension (3136,100) and (100,19)

Part a and b

Random seed 1

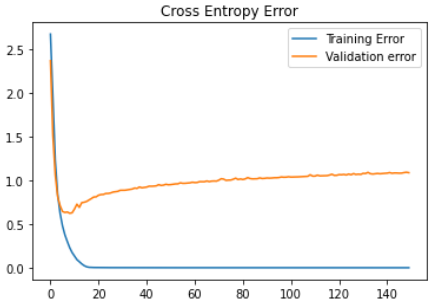
Training Loss: 0.00011539542560640257
Validation Loss: 1.2218463678553606
Model train Misclassification = 0.0
Model Test Misclassification = 0.138





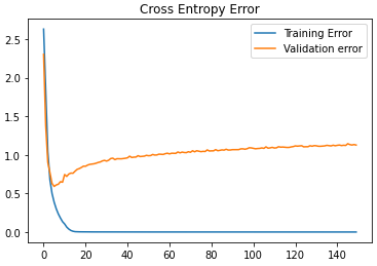
RANDOM SEED 66

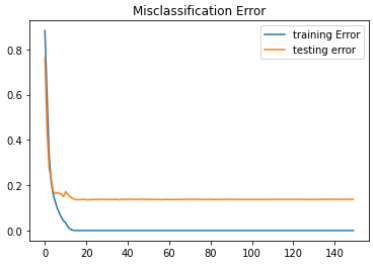
Training Loss: 0.00011501388268370647
Validation Loss: 1.085855454064099
Model train Misclassification = 0.0
Model Test Misclassification = 0.135800000000000



Random seed 88

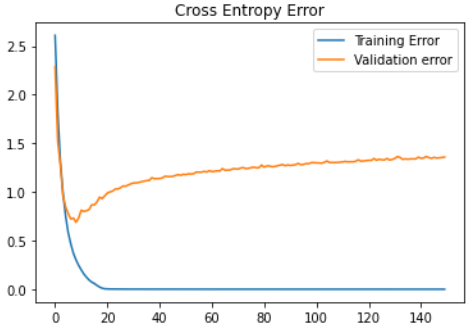
Training Loss: 0.00011537392941536382
Validation Loss: 1.126778663082677
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.13839999999999997

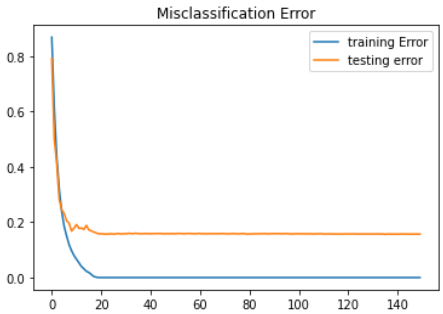




SEED 100

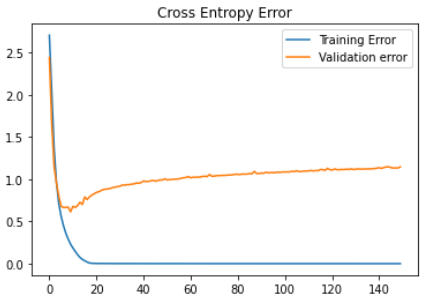
Training Loss: 0.00011469051694148 22
Validation Loss: 1.359205505270867
Model train Misclassification = 0.0
Model Test Misclassification = 0.15680000000000005

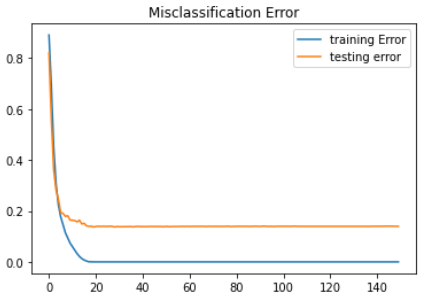




SEED 200

Training Loss: 0.00011683900458447169
Validation Loss: 1.1462519113686245
Model train Misclassification = 0.0
Model Test Misclassification = 0.13980000000000004





Random seed 66 has the least misclassification error

PART C

```
import matplotlib.pyplot as plt

params = list(model_new_cnn2.parameters())[0]
plt.figure(figsize=(8, 8))
for i in range(params.shape[0]):
    plt.subplot(2,4,i + 1) # Since we know it is a 10 x 10 grid
    x = params[i,:].detach().numpy()
    plt.imshow(x.reshape((3, 3)), cmap = "gray", interpolation = "nearest"
    plt.axis("off")
plt.subplots_adjust(wspace=0.25, hspace=0.01)
```

These depict the weights of the first convolution layer. In this case as our first convolution layer is defined as (1,8) we have 8 images displayed.

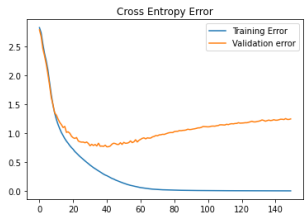
PART C

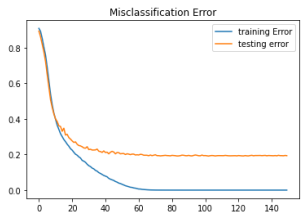
```
import matplotlib.pyplot as plt

params = list(model_new_cnn2.parameters())[0]
plt.figure(figsize=(8, 8))
for i in range(params.shape[0]):
    plt.subplot(2,4,i + 1) # Since we know it is a 10 x 10 grid
    x = params[i,:].detach().numpy()
    plt.imshow(x.reshape((3, 3)), cmap = "gray", interpolation = '
    plt.axis("off")
plt.subplots_adjust(wspace=0.25, hspace=0.01)
```

PART D

1.LR=0.01 and Momentum 0

Training Loss: 0.003128289331495762
Validation Loss: 1.246729735712147
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.19340000000000002

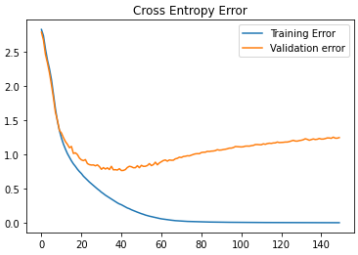


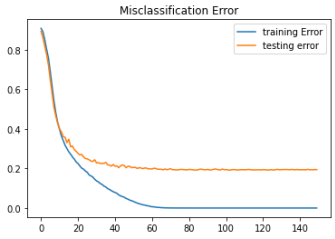


2. LR=0.01 and Momentum 0.5

Training Loss: 0.0010672599947080016
Validation Loss: 1.513383491279166

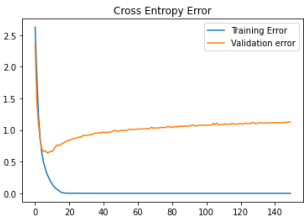
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.1906

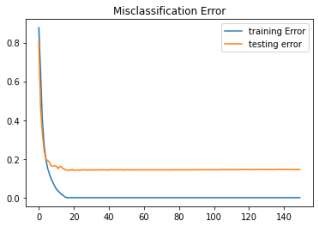




3 LR=0.01 and Momentum 0.9

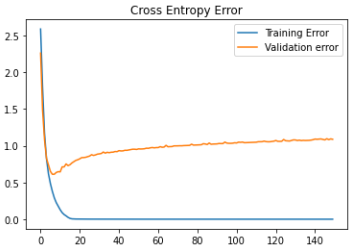
Training Loss: 0.00011623142126190942
Validation Loss: 1.1226864102516014
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.14539999999999997

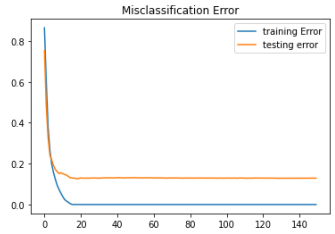




4. LR=0.1 and Momentum 0

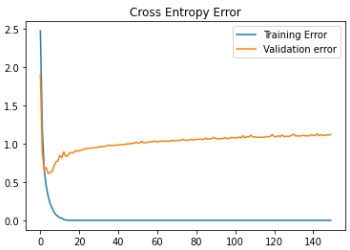
Training Loss: 0.0001152702464023605
Validation Loss: 1.0878264735583334
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.129





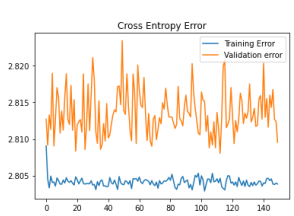
5. LR=0.1 and Momentum 0.5

Training Loss: 3.8163072186580397e-05
Validation Loss: 1.1232103755472191
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.125



6. LR=0.1 and momentum 0.9

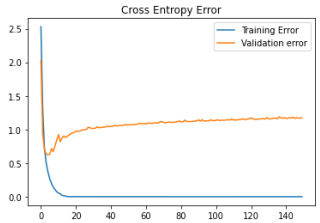
Training Loss: 2.8038607982635497
Validation Loss: 2.809546713616438
Model train MisAccuracy = 0.904
Model Test MisAccuracy = 0.9097999999999999

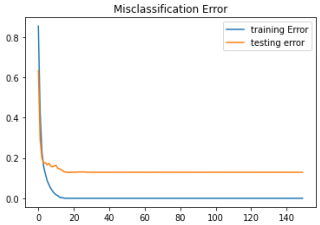


Here we can see when lr =0.1 with a higher momentum . It causes overfitting and gives a very high misclassification nd validation loss.

8. LR=0.2 and Momentum 0

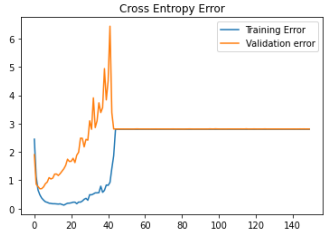
Training Loss: 3.5305813084050895e-05
Validation Loss: 1.1731375439661988
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.12919999999999998

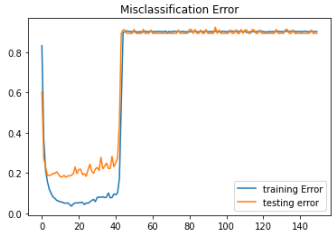




9. LR=0.2 and Momentum 0.5

Training Loss: 2.799372023773193
Validation Loss: 2.80958694105695
Model train MisAccuracy = 0.902
Model Test MisAccuracy = 0.8928



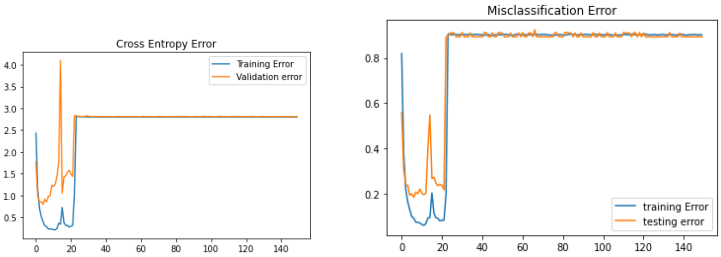


We can see a drop and the a rise , If we did early stopping before it overfit our misclassification error was 0.1894

10.LR=0.2 and momentum 0.9

Training Loss: 2.8005139446258545

Validation Loss: 2.8121326759362675
Model train MisAccuracy = 0.90065
Model Test MisAccuracy = 0.893

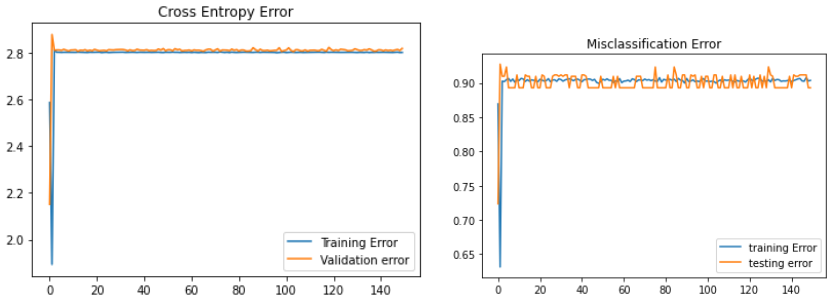


Again if we had stopped earlier our error rate is around 0.4 . However it overfits after that

11. LR=0.5 and Momentum 0.5

Training Loss: 2.8036752529144287
Validation Loss: 2.821112729941204
Model train MisAccuracy = 0.90385

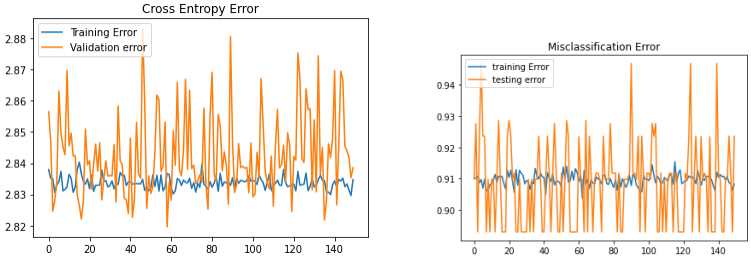
Model Test MisAccuracy = 0.893



This model has consisitently overfit. So its not a good model

12. LR=0.5 and Momentum 0.9

Training Loss: 2.834717852783203
Validation Loss: 2.838554629854336
Model train MisAccuracy = 0.90825
Model Test MisAccuracy = 0.9236



INTERPRETATION:

In this case we can see that models with lower LR have performed better. This is because in complex model its usually better to have a lower lr as model is already complex.

BEST MODEL ACCORDING TO VALIDATION ERROR IS LR=0.1 and Momentum 0.5. This is as it gave the lowest validation and misclassification test error. Misclassification error in this case is 0.125 and validation loss is 1.1232103755472191. Hence there is 87.5% accuracy in this model

MODEL2-

LE_NET5:

Lenet5 is a classic CNN architecture. two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier

Before Applying:

Our Batch Size is 128 and we also normalized the data due to presence of nan.

MODEL ARCHITECTURE:

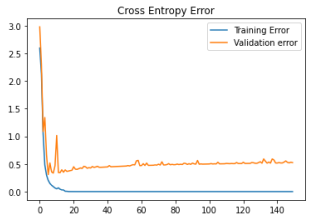
```
class LeNet5(Module):
    def __init__(self, num_classes=19):
        super(LeNet5, self).__init__()
        self.layer1 = Sequential(
            Conv2d(1, 6, kernel_size=5, stride=1, padding=0),
            BatchNorm2d(6),
            ReLU(),
            MaxPool2d(kernel_size = 2, stride = 2))
        self.layer2 = Sequential(
            Conv2d(6, 16, kernel_size=5, stride=1, padding=0),
            BatchNorm2d(16),
            ReLU(),
            MaxPool2d(kernel_size = 2, stride = 2))
        self.fc = Linear(704, 120)
        self.relu = ReLU()
        self.fc1 = Linear(120, 84)
        self.relu1 = ReLU()
        self.fc2 = Linear(84, num_classes)
        self.drop=nn.Dropout(0.1)

    def forward(self, x):
        out = self.layer1(x)
        # print(out.shape)
        out = self.layer2(out)
        # print("layer2",out.shape)
        out = out.reshape(out.size(0), -1)
        # print("reshape",out.shape)
        out=self.drop(out)
        out = self.fc(out)
        out = self.relu(out)
        out = self.fc1(out)
        out = self.relu1(out)
        out = self.fc2(out)
        return out
```

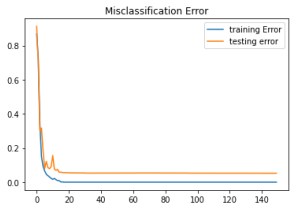
Hence we have 2 Convulution Layers. 2 Linear layers. 1 output layer. Besides this we are also applying relu , dropout and maxpool Batchnorm2d

PART A AND B

RANDOM.SEED(1)

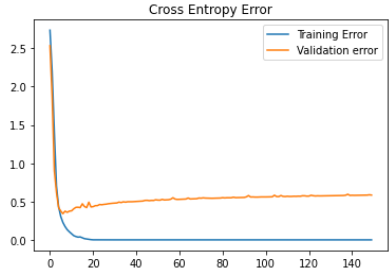
Training Loss: 7.825620670677857e-06
Validation Loss: 0.5246614467352628
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.052200000000000024

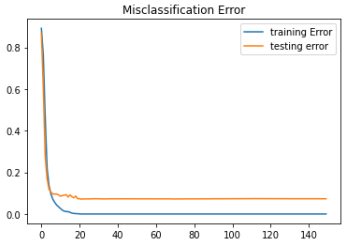




RANDOM.SEED(66)

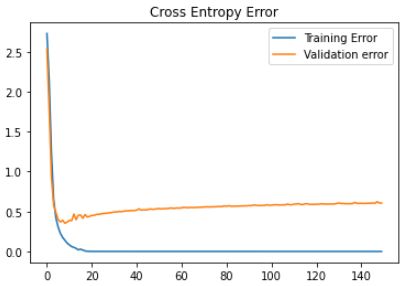
Training Loss: 1.9716167952753947e-05
Validation Loss: 0.5845402058512031
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.07320000000000004

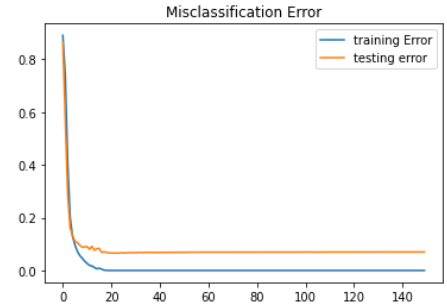




RANDOM.SEED(88)

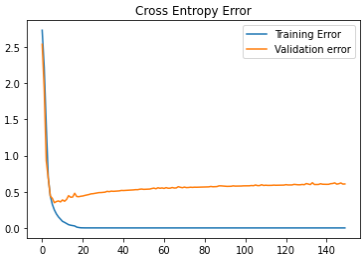
Training Loss: 2.218370843838784e-05
Validation Loss: 0.6048760045305244
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.06999999999999995

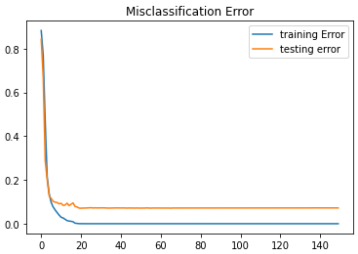




RANDOM SEED(100)

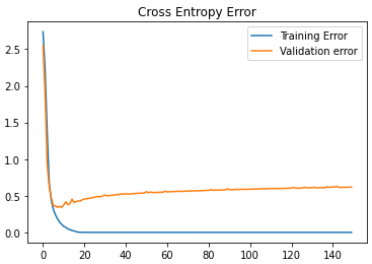
Training Cross entropy Loss: 2.3770248665459803e-05
Validation Cross Entropy Loss: 0.6079011502760335
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.07199999999999995

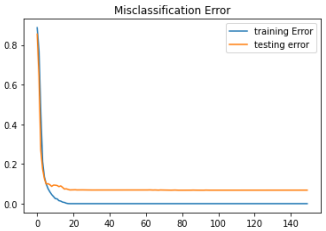




RANDOM.SEED(200)

hTraining Loss: 2.6972041727640317e-05
Validation Loss: 0.6174310335109634
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.06820000000000004





Hence we select Random seed 1 due to lowest Validation Loss.Its test error is 0.052200000000000024

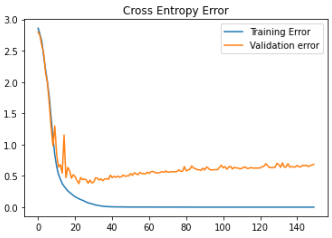
PART C:

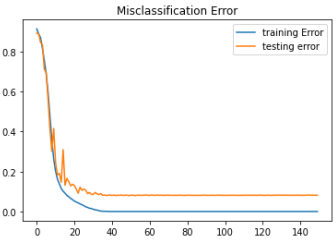
```
] import matplotlib.pyplot as plt

  params = list(model_new_cnn3.parameters())[0]
  plt.figure(figsize=(8, 8))
  for i in range(params.shape[0]):
      plt.subplot(2,4,i + 1) # Since we know it is a 10 x 10 grid
      x = params[i,:].detach().numpy()
      plt.imshow(x.reshape((5, 5)), cmap = "gray", interpolation = "nearest")
      plt.axis("off")
  plt.subplots_adjust(wspace=0.25, hspace=0.01)
```

PART D

1.LR=0.01 and Momentum 0:

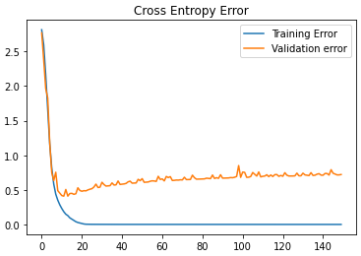
Training Loss: 0.0002416361185516658
Validation Loss: 0.683994073420763
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.08099999999999996

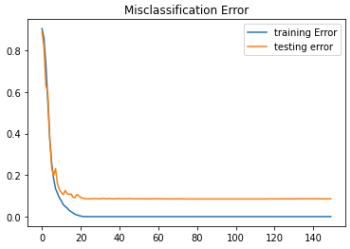




2. LR=0.01 and Momentum 0.5:

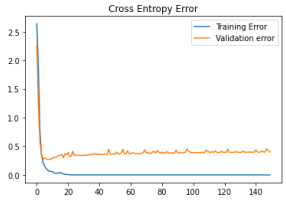
Training Loss: 6.761906161914715e-05
Validation Loss: 0.7217757657170296
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.08579999999999999

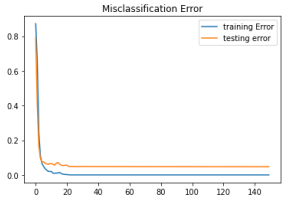




3. LR=0.01 and Momentum 0.9:

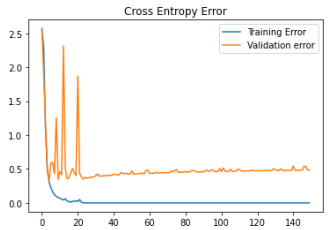
Training Loss: 5.215314724343217e-06
Validation Loss: 0.3998927651264239
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.04720000000000002

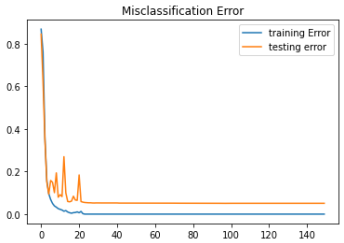




4.LR=0.1 and Momentum 0:

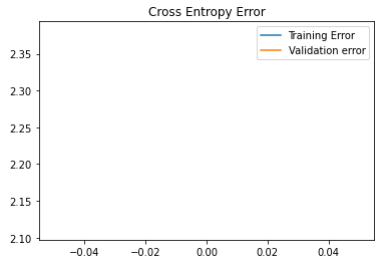
Training Loss: 7.836332385007263e-06
Validation Loss: 0.48387249365914614
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.050799999999999956

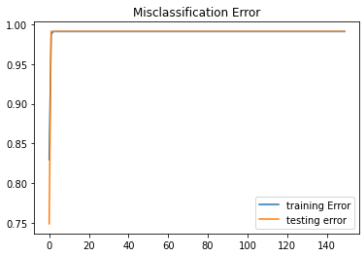




5. LR=0.1 and Momentum 0.5:

Training Loss: nan
Validation Loss: nan
Model train MisAccuracy = 0.991
Model Test MisAccuracy = 0.9912

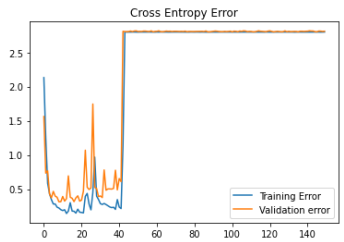


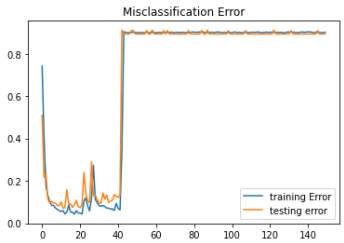


This model included a lot of nan value. The reason could be due that lr is too large. Hence it cant calculate validation and traing cross entropy error

6. LR=0.1 and Momentum 0.9:

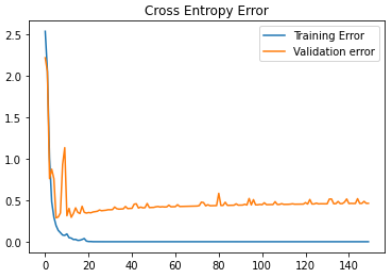
Training Loss: 2.7994877426487625
Validation Loss: 2.8106164753437044
Model train MisAccuracy = 0.90145
Model Test MisAccuracy = 0.893

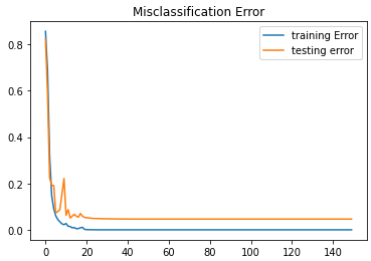




7 LR=0.2 AND Momentum=0

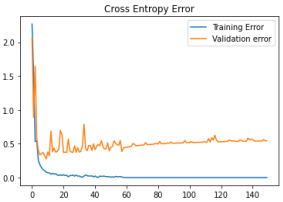
Training Loss: 4.083562783160574e-06
Validation Loss: 0.46402543497169974
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.04620000000000002

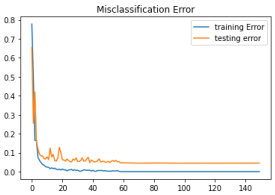




8.LR=0.2 and Momentum 0.5:

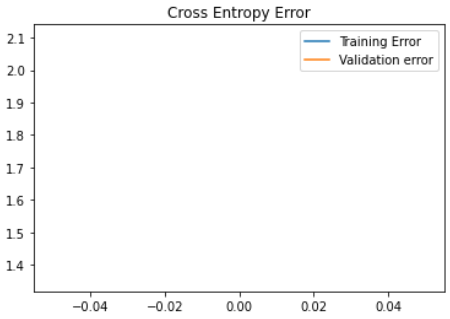
Training Loss: 1.398641006131344e-06
Validation Loss: 0.5438249861821532
Model train MisAccuracy = 0.0
Model Test MisAccuracy = 0.04459999999999997

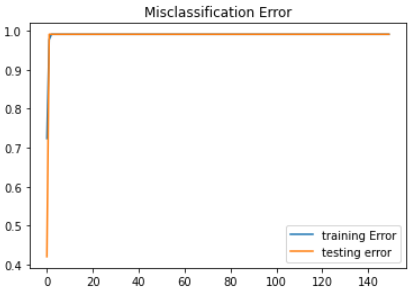




9. LR=0.2 and Momentum 0.9:

Training Loss: nan
Validation Loss: nan
Model train MisAccuracy = 0.991
Model Test MisAccuracy = 0.9912



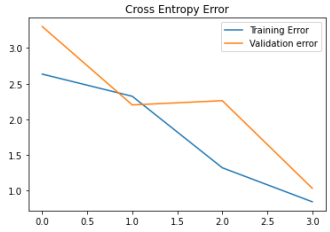


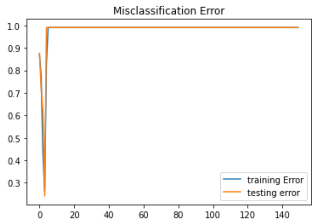
Again LR is too large

10.LR=0.5 and Momentum 0:

Training Loss: nan
Validation Loss: nan

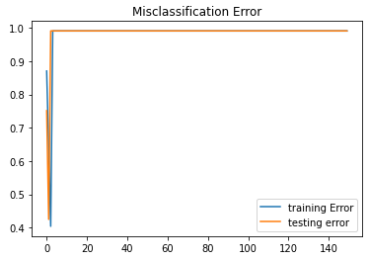
Model train MisAccuracy = 0.991
Model Test MisAccuracy = 0.9912





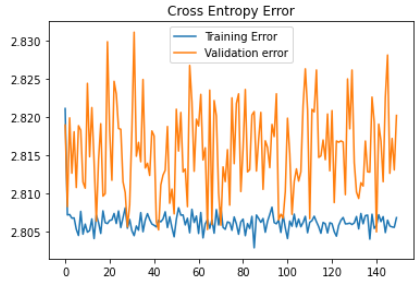
12.LR=0.5 and Momentum 0.5:

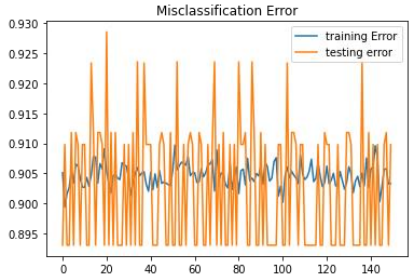
Training Loss: nan
Validation Loss: nan
Model train MisAccuracy = 0.991
Model Test MisAccuracy = 0.9912



12. LR=0.5 and Momentum 0.9:

Training Loss: 2.806874821899803
Validation Loss: 2.8202144682407377
Model train MisAccuracy = 0.9033
Model Test MisAccuracy = 0.9097999999999999





INTERPRETATION:

Here we can see model works very badly in cases of higher lr. The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Here our high lr led to unstable training and hence failure to converge in some cases.

However for lower learning rate accuracy Is very high.

OUR BEST MODEL IN THIS CASE IS LR=0.2 AND MOMENTUM 0 . Validation Loss in this case is 0.46402543497169974 and Genralizable Test MisAccuracy = 0.04620000000000002. This shows there is approximately 95.38% accuracy using LeNet5 architecture.

Comparison to MNIST test errors:

We see that MNIST did generally have lower test errors than Question 6 and 7. Best Models in our MNIST has a accuracy of 98% whereas here is 95.3%. This is because previously we were predicting just one number .However in second case we are making neural network recognise two number and then also recognise that it's the sum.Hence error is higher in the second case.

We cant have test error lower 1% . This is because models leave some room for generalizability.

REFERENCES:

https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/