Deep Learning Report

Programming Assignment 1

Ayush Abrol
B20AI052

Question 01

Aim: Implement a neural network and utilize the CIFAR-10 dataset for the analysis.

1. Utilize various activation functions like sigmoid, tanh and critique the performance in each case.

2. Increase the depth of the given network by adding more Fully-Connected layers till the point you encounter the vanishing gradient problem. With the help of the results, mention how to identify it.

3. Suggest and implement methods to overcome the above problem.

Procedure:

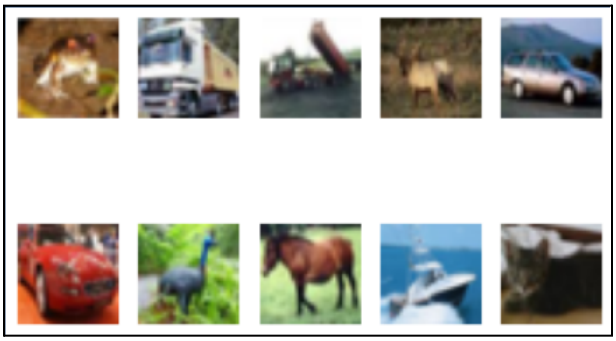
- All the necessary dependencies and libraries were taken care of initially like:
  - numpy, pandas, matplotlib for data manipulation and visualization.
  - Torch, torchvision, torch.nn, PIL for for implementing Deep Learning architectures where necessary using PyTocrch.
- Loading of the training and the testing CIFAR10 dataset and normalizing it using torchvision's transforms module. 50k training samples and 10k testing samples are loaded and converted into Torch format Tensors.

```
Train data shape:  (50000, 32, 32, 3)
Test data shape:  (10000, 32, 32, 3)
Train data labels shape:  (50000,)
Test data labels shape:  (10000,)
```

- Converted the shape into PyTorch's input format using permute(0, 3, 1, 2).
- We had the following 10 classes.

```
classes = ['plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

- Visualizing few images of the dataset.



- Implemented the **CNN architecture** using torch.nn with 3 convolutional layers where:
    - Conv1: in_channels = 3, out_channels = 32
    - Conv2: in_channels = 32, out_channels = 64
    - Conv3: in_channels = 64, out_channels = 128
    - Then added the fully connected hidden layers to network where the number of layers and the number of neurons in each layer are taken as model parameters.
    - Also, the activation function is also passed as a parameter to the model.
    - Input to the first fully connected hidden layer is 128*4*4 and the sizes of further layers is passed corresponding to the number of hidden layers created.
    - Output layer is created at last of size 10 on which the Softmax activation is applied for getting the required outputs.
- Created object of CNN class using ReLU, Sigmoid and Tanh activation functions with 3 hidden layers with sizes [512, 256, 128] output layer of size 10.

```
CNN(
  (activation_function): ReLU()
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc_block): Sequential(
    (fc0): Linear(in_features=2048, out_features=512, bias=True)
    (activation0): ReLU()
    (fc1): Linear(in_features=512, out_features=256, bias=True)
    (activation1): ReLU()
    (fc2): Linear(in_features=256, out_features=128, bias=True)
    (activation2): ReLU()
    (fc3): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

```
CNN(
  (activation_function): Sigmoid()
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc_block): Sequential(
    (fc0): Linear(in_features=2048, out_features=512, bias=True)
    (activation0): Sigmoid()
    (fc1): Linear(in_features=512, out_features=256, bias=True)
    (activation1): Sigmoid()
    (fc2): Linear(in_features=256, out_features=128, bias=True)
    (activation2): Sigmoid()
    (fc3): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

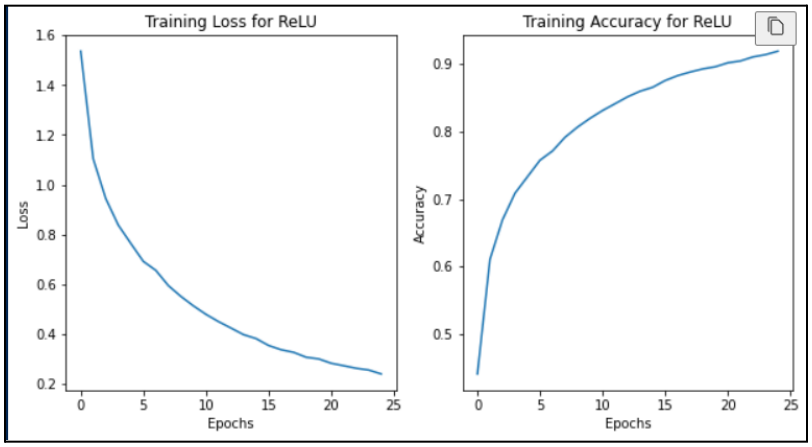
```
CNN(
  (activation_function): Tanh()
  (conv1): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc_block): Sequential(
    (fc0): Linear(in_features=2048, out_features=512, bias=True)
    (activation0): Tanh()
    (fc1): Linear(in_features=512, out_features=256, bias=True)
    (activation1): Tanh()
    (fc2): Linear(in_features=256, out_features=128, bias=True)
    (activation2): Tanh()
    (fc3): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

- Used the Cross Entropy loss as the loss function and Adam optimizer with learning rate = 1e-3.

- Trained the model for ReLU activation with 25 epochs and calculated the training loss, training accuracy with epoch.

```
Epoch: 1 Training Loss:  1.5362005803896033      Training Accuracy:  0.44118
Epoch: 2 Training Loss:  1.1037224537271368      Training Accuracy:  0.61062
Epoch: 3 Training Loss:  0.9448575418623512      Training Accuracy:  0.6691
Epoch: 4 Training Loss:  0.8382054520080157      Training Accuracy:  0.70862
Epoch: 5 Training Loss:  0.7647262541839229      Training Accuracy:  0.73312
Epoch: 6 Training Loss:  0.6925711412259075      Training Accuracy:  0.75756
Epoch: 7 Training Loss:  0.6564889127945961      Training Accuracy:  0.77122
Epoch: 8 Training Loss:  0.5953130887444976      Training Accuracy:  0.7915
Epoch: 9 Training Loss:  0.5520661664588372      Training Accuracy:  0.8066
Epoch: 10 Training Loss:  0.5140521828170932     Training Accuracy:  0.81956
Epoch: 11 Training Loss:  0.4801609010037864     Training Accuracy:  0.83112
Epoch: 12 Training Loss:  0.45099517329574546    Training Accuracy:  0.8412
Epoch: 13 Training Loss:  0.4250290161737091     Training Accuracy:  0.8514
Epoch: 14 Training Loss:  0.3986660727988119     Training Accuracy:  0.85968
Epoch: 15 Training Loss:  0.3828735051252653     Training Accuracy:  0.86546
Epoch: 16 Training Loss:  0.3551832173410279     Training Accuracy:  0.8755
Epoch: 17 Training Loss:  0.33802910686453896    Training Accuracy:  0.88272
Epoch: 18 Training Loss:  0.3275901391683027     Training Accuracy:  0.88806
Epoch: 19 Training Loss:  0.30794802957864675    Training Accuracy:  0.89266
Epoch: 20 Training Loss:  0.3012029437152931     Training Accuracy:  0.89584
Epoch: 21 Training Loss:  0.28379474629831436    Training Accuracy:  0.90168
Epoch: 22 Training Loss:  0.2738388474563808     Training Accuracy:  0.90438
Epoch: 23 Training Loss:  0.2636131509719297     Training Accuracy:  0.91034
Epoch: 24 Training Loss:  0.256544516481402      Training Accuracy:  0.91372
Epoch: 25 Training Loss:  0.24078244441534247    Training Accuracy:  0.91888
Training Complete for ReLU.
```

- Plot for training loss and accuracy for ReLU.



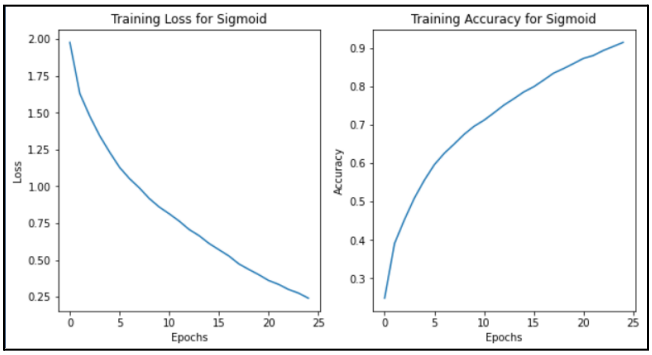
- Then, tested the model with ReLU and got the following results.

```
Testing Complete for ReLU.
Test Loss:  0.7992848233331608   Test Accuracy:  0.7525712025316456
```

- Trained the model for Sigmoid activation with 25 epochs and calculated the training loss, training accuracy with epoch.

```
Epoch: 1 Training Loss:  1.9774655884184191      Training Accuracy:  0.24876
Epoch: 2 Training Loss:  1.6305725056192149      Training Accuracy:  0.39134
Epoch: 3 Training Loss:  1.4773083257553217      Training Accuracy:  0.45346
Epoch: 4 Training Loss:  1.3439248467955138      Training Accuracy:  0.50942
Epoch: 5 Training Loss:  1.232689294516278       Training Accuracy:  0.55588
Epoch: 6 Training Loss:  1.1293360011656877      Training Accuracy:  0.59604
Epoch: 7 Training Loss:  1.0527494909513333      Training Accuracy:  0.6257
Epoch: 8 Training Loss:  0.9892077064880019      Training Accuracy:  0.64994
Epoch: 9 Training Loss:  0.9171169249298018      Training Accuracy:  0.67512
Epoch: 10 Training Loss:  0.8596349367705147     Training Accuracy:  0.69606
Epoch: 11 Training Loss:  0.8140531141129906     Training Accuracy:  0.71186
Epoch: 12 Training Loss:  0.7645191626475595     Training Accuracy:  0.73114
Epoch: 13 Training Loss:  0.7074978338635486     Training Accuracy:  0.75114
Epoch: 14 Training Loss:  0.6657679904147488     Training Accuracy:  0.76778
Epoch: 15 Training Loss:  0.6129927313541208     Training Accuracy:  0.78546
Epoch: 16 Training Loss:  0.5701305190163195     Training Accuracy:  0.79892
Epoch: 17 Training Loss:  0.5277554985050046     Training Accuracy:  0.81628
Epoch: 18 Training Loss:  0.4741922309026694     Training Accuracy:  0.8343
Epoch: 19 Training Loss:  0.436499309051982      Training Accuracy:  0.84626
Epoch: 20 Training Loss:  0.40096007081706203    Training Accuracy:  0.85922
Epoch: 21 Training Loss:  0.3612592826261545     Training Accuracy:  0.87274
Epoch: 22 Training Loss:  0.33498828193110886    Training Accuracy:  0.8803
Epoch: 23 Training Loss:  0.30055201320392094    Training Accuracy:  0.89328
Epoch: 24 Training Loss:  0.2758623786137232     Training Accuracy:  0.90378
Epoch: 25 Training Loss:  0.24115315054917275    Training Accuracy:  0.91442
Training Complete for Sigmoid.
```

- Plot for training loss and accuracy for Sigmoid.



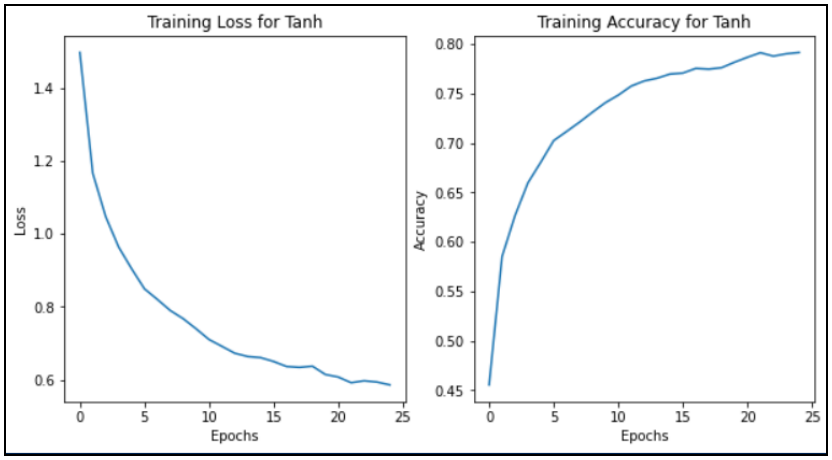
- Then, tested the model with ReLU and got the following results.

```
Testing Complete for Sigmoid.
Test Loss:  0.9906675415703013   Test Accuracy:  0.7201344936708861
```

- Trained the model for Tanh activation with 25 epochs and calculated the training loss, training accuracy with epoch.

```
Epoch: 1 Training Loss:  1.4962909901538468      Training Accuracy:  0.45532
Epoch: 2 Training Loss:  1.1658362205071218      Training Accuracy:  0.58516
Epoch: 3 Training Loss:  1.047083021277357       Training Accuracy:  0.6265
Epoch: 4 Training Loss:  0.9639698352350299      Training Accuracy:  0.6597
Epoch: 5 Training Loss:  0.9047298795731781      Training Accuracy:  0.68072
Epoch: 6 Training Loss:  0.8494244829163222      Training Accuracy:  0.70258
Epoch: 7 Training Loss:  0.8203854801709695      Training Accuracy:  0.7118
Epoch: 8 Training Loss:  0.7900081661046313      Training Accuracy:  0.72122
Epoch: 9 Training Loss:  0.7676092810033227      Training Accuracy:  0.7313
Epoch: 10 Training Loss:  0.7399164097540823     Training Accuracy:  0.74092
Epoch: 11 Training Loss:  0.710596130174749      Training Accuracy:  0.74856
Epoch: 12 Training Loss:  0.6918770491009782     Training Accuracy:  0.75772
Epoch: 13 Training Loss:  0.6729594694683924     Training Accuracy:  0.76288
Epoch: 14 Training Loss:  0.6640662709465417     Training Accuracy:  0.76566
Epoch: 15 Training Loss:  0.6609873633707881     Training Accuracy:  0.76988
Epoch: 16 Training Loss:  0.6504805352529297     Training Accuracy:  0.7709
Epoch: 17 Training Loss:  0.6368091982953689     Training Accuracy:  0.77562
Epoch: 18 Training Loss:  0.6344937872703728     Training Accuracy:  0.77484
Epoch: 19 Training Loss:  0.6376625134816865     Training Accuracy:  0.77632
Epoch: 20 Training Loss:  0.6150012908841643     Training Accuracy:  0.7819
Epoch: 21 Training Loss:  0.6079372813177231     Training Accuracy:  0.78688
Epoch: 22 Training Loss:  0.5923699174085846     Training Accuracy:  0.79148
Epoch: 23 Training Loss:  0.5976631495044055     Training Accuracy:  0.788
Epoch: 24 Training Loss:  0.5942446393582522     Training Accuracy:  0.79038
Epoch: 25 Training Loss:  0.5863842541909279     Training Accuracy:  0.7917
Training Complete for Tanh.
```

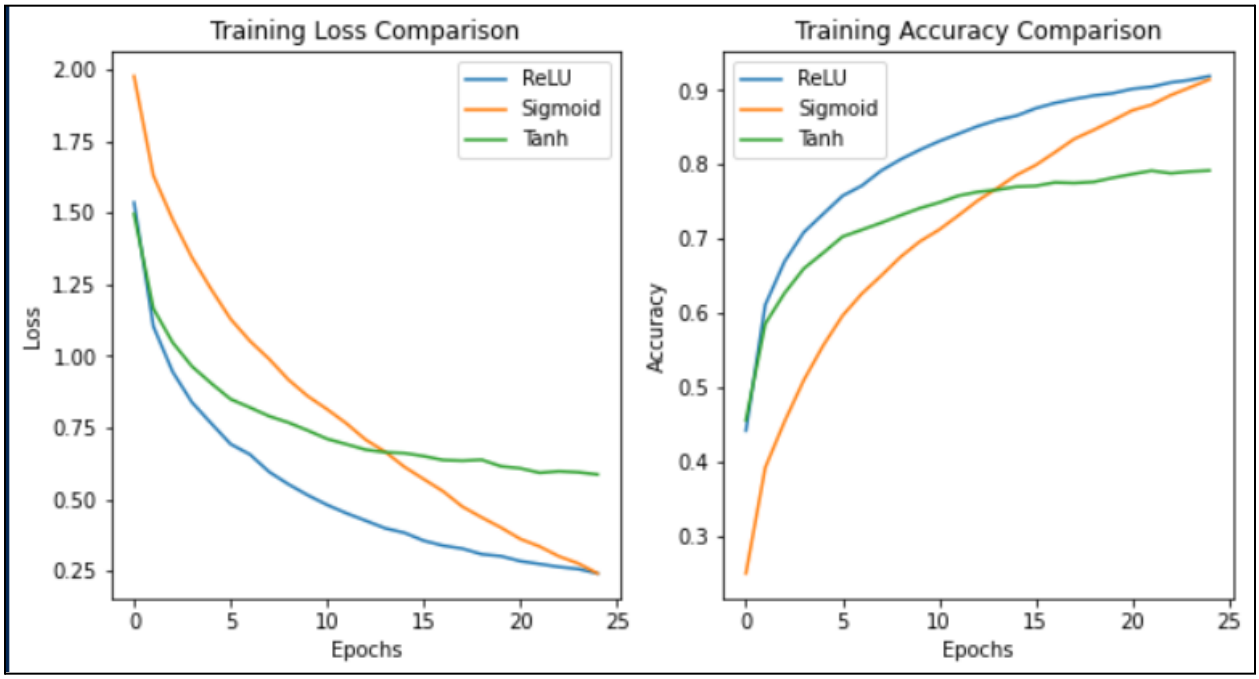
- Plot for training loss and accuracy for Tanh.



- Then, tested the model with Tanh and got the following results.

```
Testing Complete for Tanh.
Test Loss:  1.2179340682452238   Test Accuracy:  0.6220332278481012
```

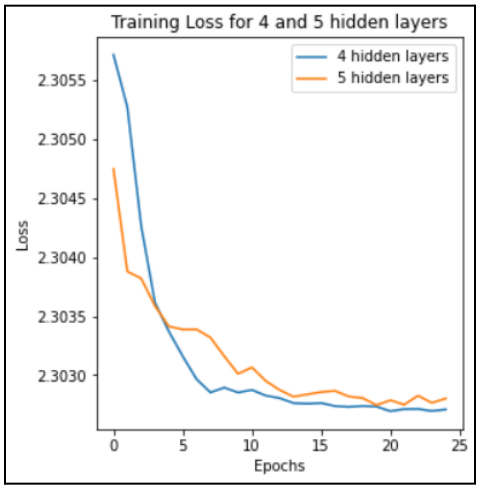
- Comparison plot between the losses and accuracies of the three activation functions.



- Increasing the depth of the given network (Sigmoid or tanh) by adding more Fully-Connected layers till the point we encounter the **vanishing gradient problem.**
- **Note**: We can't use ReLU here as it won't lead to vanishing gradient problem because it is a non-saturating function which means that it doesn't saturate to 0 or 1.
- Then, I found the number of layers of for the **Sigmoid and the Tanh functions** where the vanishing gradient problem occurs by consistently increasing the number of hidden layers.

```
# Increasing the number of hidden layers for Sigmoid activation function
model_list_fc_layers = []
model_list_fc_layers.append(CNN(nn.Sigmoid(), 4, [512, 256, 128, 64], 10).to(device))
model_list_fc_layers.append(CNN(nn.Sigmoid(), 5, [512, 256, 128, 64, 32], 10).to(device))
model_list_fc_layers.append(CNN(nn.Sigmoid(), 6, [512, 256, 128, 64, 32, 16], 10).to(device))
model_list_fc_layers.append(CNN(nn.Sigmoid(), 7, [512, 384, 256, 128, 64, 32, 16], 10).to(device))
```

- Therefore, at 5 fully connected layers for Sigmoid function, we will encounter the vanishing gradient problem because curve of Sigmoid becomes almost flat after 5 layers and therefore we infer that difference between the losses of different epochs is very very less and hence the gradient is vanishing.
- Plot for training loss for 4 and 5 hidden layers for Sigmoid.



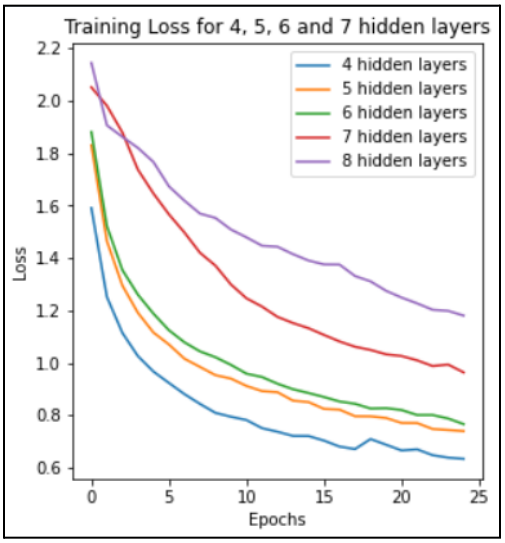
- As, the difference between the loss from epoch 10 to 25 is almost 0 for the 5 hidden layers, hence vanishing gradient is occuring.

- Similarily, for the Tanh function, I trained different models with increasing number of hidden layers.

```
model_list_fc_layers_tanh = []
model_list_fc_layers_tanh.append(CNN(nn.Tanh(), 4, [512, 256, 128, 64], 10).to(device))
model_list_fc_layers_tanh.append(CNN(nn.Tanh(), 5, [512, 256, 128, 64, 32], 10).to(device))
model_list_fc_layers_tanh.append(CNN(nn.Tanh(), 6, [512, 256, 128, 64, 32, 16], 10).to(device))
model_list_fc_layers_tanh.append(CNN(nn.Tanh(), 7, [512, 384, 256, 128, 64, 32, 16], 10).to(device))
```

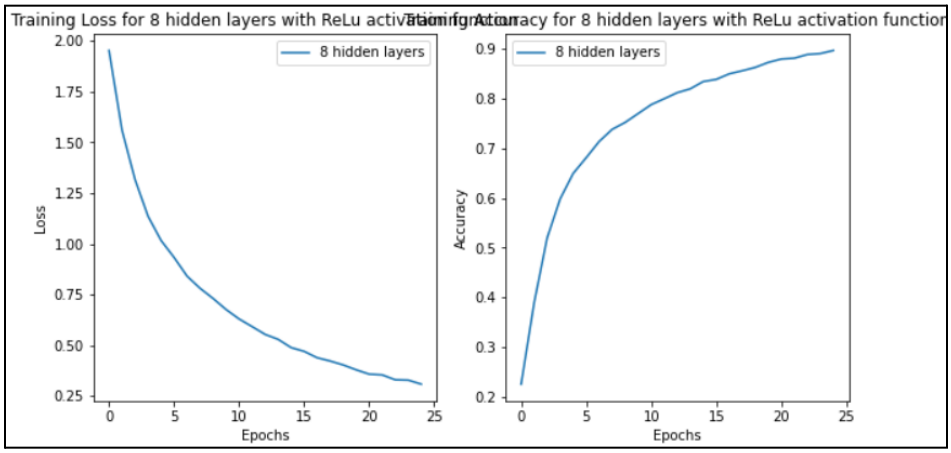
- Therefore, at 8 fully connected layers for Tanh function, we will encounter the vanishing gradient problem because curve of Tanh becomes almost flat after 8 layers and therefore we infer that difference between the losses of different epochs is very very less and hence the gradient is vanishing.

- Plot for training loss for 4, 5, 6, 7 and 8 hidden layers for Tanh.

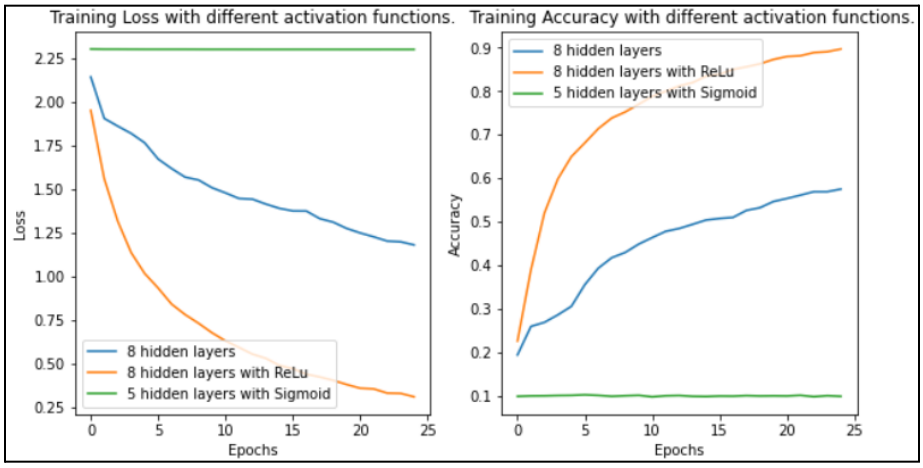


- As the curve for the 8 hidden layers is tending towards almost zero loss difference after 15 epochs, we can infer from here that vanishing gradient problem is occurring in this case.
- Therefore, for the Sigmoid and Tanh activation functions, vanishing gradient problem is occurring because these functions saturate between 0 and 1 and gradient keeps on becoming very small values with increasing number of values.

- As a **method to overcome the vanishing gradient problem**, I trained the model with ReLU activation function with upto 8 hidden layers and got the following plot.



- We are using ReLU activation function here as it is a non-saturating function which means that it doesn't saturate to 0 or 1 and even with increasing number of layers, it doesn't lead to vanishing gradient problem.
- Comparison between the 8 layer network with ReLU activation function, 8 layer network with tanh activation function and 5 hidden layer network with sigmoid activation function.



END OF QUESTION 1.

Question 02

Aim: Implement a neural network on the **Gurmukhi dataset** and implement the following regularizationtechniques from scratch:

1. L-1 regularization

2. L-2 regularization

3. Dropout

Compare the performance of the above techniques and mention reasons to support your answer. Also, implement gradient checking (from scratch) to verify the values of gradients during backpropagation.

Procedure:

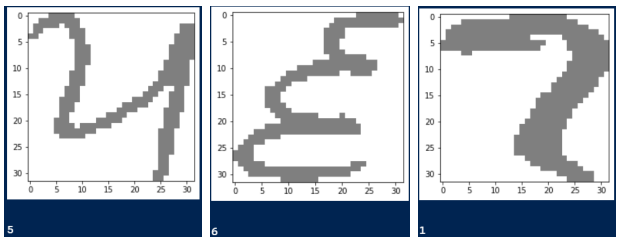
- Loading of the training and the testing Gurmukhi dataset and normalizing it using torchvision's transforms module. 1000training samples and 178 testing samples are loaded and converted into Torch format Tensors.

```
X_train_gur shape:  torch.Size([1000, 3, 32, 32])
y_train_gur shape:  torch.Size([1000])
X_test_gur shape:  torch.Size([178, 3, 32, 32])
y_test_gur shape:  torch.Size([178])
```

- Converted the shape into PyTorch's input format using permute(0, 3, 1, 2).
- We had the following 10 classes.

```
classes_gur = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

- Visualizing few images of the dataset.

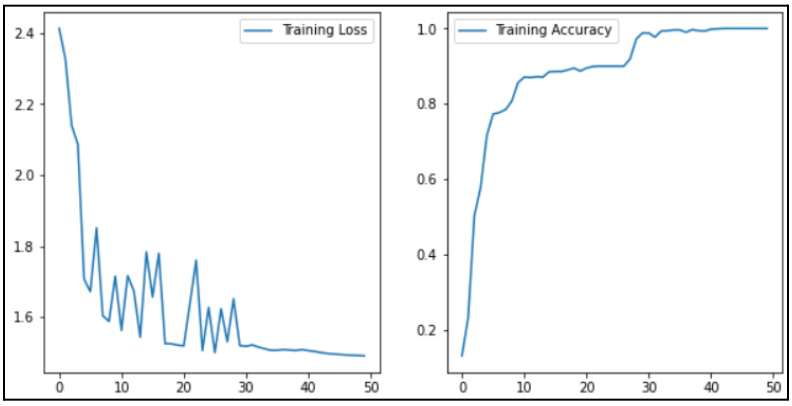


- Implemented the **CNN architecture (CNN_Gurmukhi)** using torch.nn with 3 convolutional layers where:
    - Conv1: in_channels = 3, out_channels = 32
    - Conv2: in_channels = 32, out_channels = 64
    - Conv3: in_channels = 64, out_channels = 128
    - Then added the fully connected hidden layers to network where the number of layers and the number of neurons in each layer are taken as model parameters.
    - Also, the activation function, l1_factor(lambda_1), l2_factor(lambda_2) is also passed as a parameter to the model.
    - Input to the first fully connected hidden layer is 128*4*4 and the sizes of further layers are [512, 256].
    - Output layer is created at last of size 10 on which the Softmax activation is applied for getting the required outputs.
    - Layer model

```
def forward(self, x):
    x = self.conv1(x)
    x = self.activation_function(x)
    x = self.dropout(x)
    x = self.pool(x)
    x = self.conv2(x)
    x = self.activation_function(x)
    x = self.dropout(x)
    x = self.pool(x)
    x = self.conv3(x)
    x = self.activation_function(x)
    x = self.dropout(x)
    x = self.pool(x)
    x = x.view(-1, 128 * 4 * 4)
    x = self.fc1(x)
    x = self.activation_function(x)
    x = self.dropout(x)
    x = self.fc2(x)
    x = self.activation_function(x)
    x = self.dropout(x)
    x = self.fc3(x)
    x = nn.Softmax(dim=1)(x)
    return x
```

- Used the ReLU activation function for defining the model and used the Cross Entropy loss as the loss function and Adam optimizer with learning rate 1e-3.

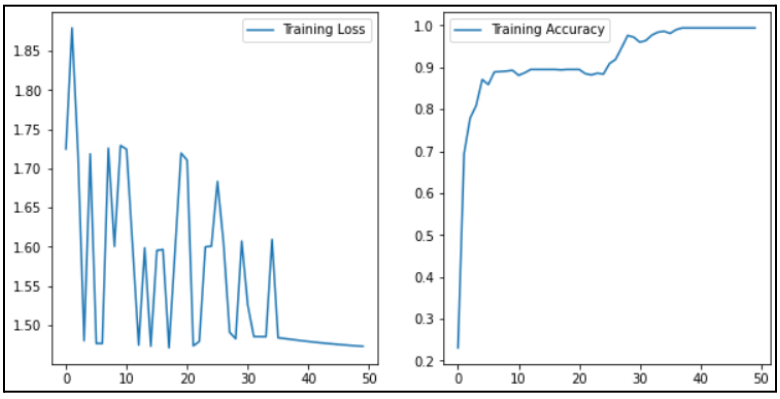
- Trained the model with l1_factor = 0.0001 and implemented the norm_1 from scratch for L1 regularization. Used 50 epochs.
- Plot for loss and accuracy curve for training data with L1 regularization.



- Testing the model after applying L1 regularization.

```
Test Loss of the model on the test images: 1.5459268093109l3
Test Accuracy of the model on the test images: 95.50561797752809 %
Testing complete with L1 regularization.
```

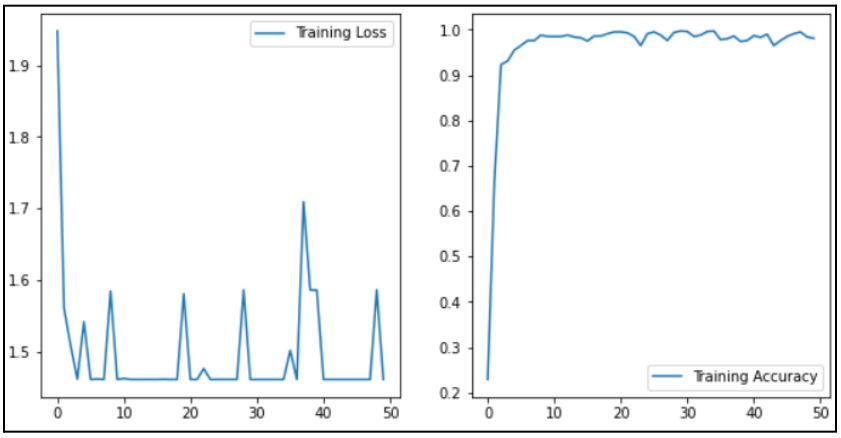
- Trained the model with l2_factor = 0.0001 and implemented the norm_2 from scratch for L2 regularization. Used 50 epochs.
- Plot for loss and accuracy curve for training data with L2 regularization.



- Testing the model after applying L1 regularization.

```
Test Loss of the model on the test images: 1.4728970527648926
Test Accuracy of the model on the test images: 97.19101123595506 %
Testing complete with L2 regularization.
```

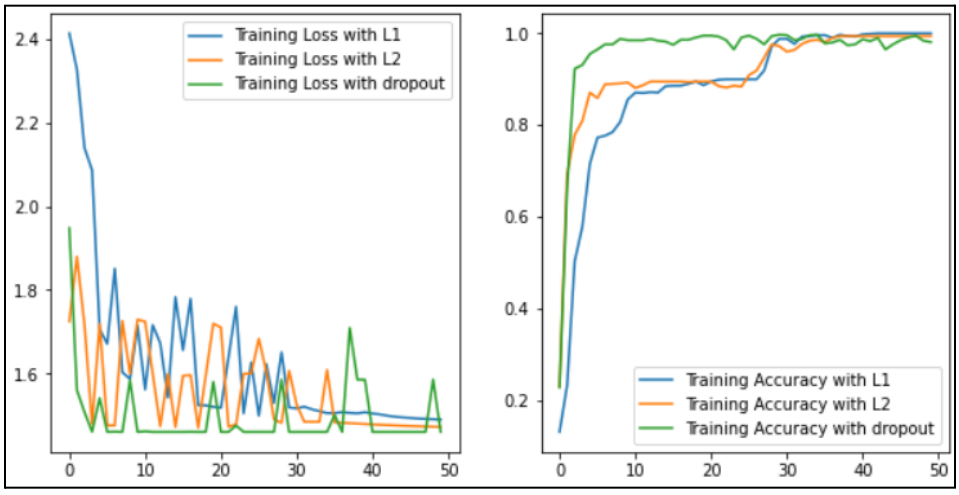
- Trained the model with l1_factor = 0, l2_factor = 0 and dropout probability = 0.2 and implemented the Dropout using PyTorch's nn.Dropout(). Used 50 epochs.
- Plot for loss and accuracy curve for training data with Dropout with dropout probability = 0.2.



- Testing the model after applying Dropout.

```
Test Loss of the model on the test images: 1.4611506462097168
Test Accuracy of the model on the test images: 93.82022471910112 %
Testing complete with dropout.
```

- Comparison plots for the loss and accuracy curve for training data with L1, L2 regularization and dropout.



- Then, implement gradient checking to verify the values of gradients during backpropagation. Set the values:
    - L1_factor = 0.0001
    - L2_factor = 0.0001
    - Dropout_probability = 0.2
- Got the outputs as:

```
tensor([ 0.0037,  0.0037, -0.0090,  0.0009,  0.0038, -0.0122,  0.0043,  0.0008,
        -0.0024,  0.0064], device='cuda:0')
```

END OF QUESTION 2.