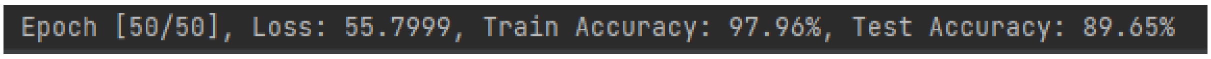
CNN able to distinguish amongst 10 different classes, also known as multi-class classification. The appropriate loss function to use in this case is categorical cross entropy,
which is used in models where the output is a probability ranging between 0 and 1. Its loss increases as prediction diverges from the ground truth, therefore a probability predicted for example of 0.01 with actual value 1 would result in a large loss value. Since this assignment encourages the use of Pytorch, the loss function “CrossEntropyLoss()” is used, which also uses the activation function softmax.

b)

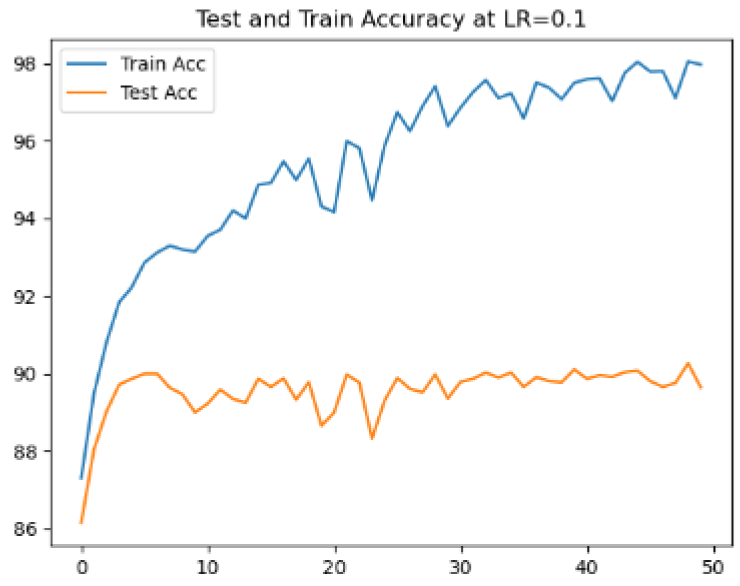
Graph and data are shown below.

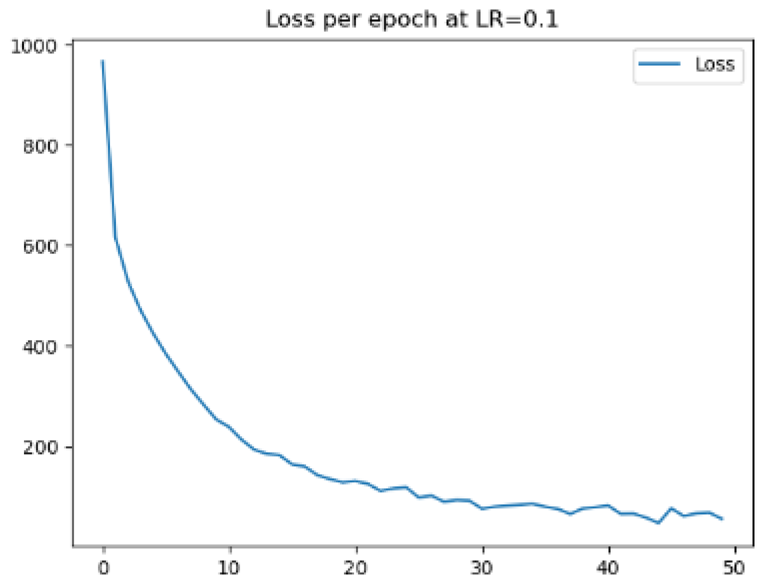
It can be observed that as the model is trained, the loss decreases over the epochs as the accuracy of both the testing and training increase. During the first couple of iterations the loss is reduced greatly after which it plenteous slightly. This is due to the cross entropy

$$H(y,p) = -\sum_i y_i \log(p_i)$$

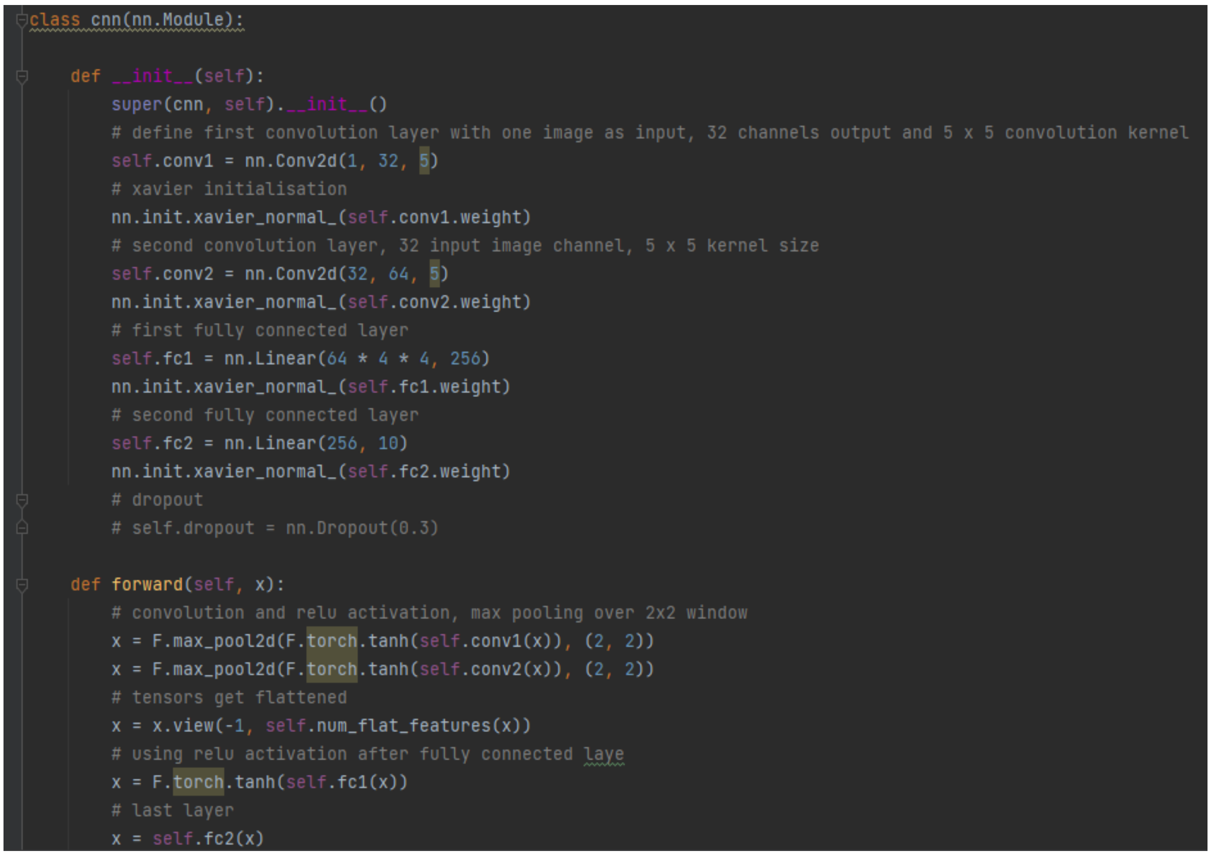


```
Epoch [50/50], Loss: 55.7999, Train Accuracy: 97.96%, Test Accuracy: 89.65%
```





Architecture of the CNN:



```
class cnn(nn.Module):

    def __init__(self):
        super(cnn, self).__init__()
        # define first convolution layer with one image as input, 32 channels output and 5 x 5 convolution kernel
        self.conv1 = nn.Conv2d(1, 32, 5)
        # xavier initialisation
        nn.init.xavier_normal_(self.conv1.weight)
        # second convolution layer, 32 input image channel, 5 x 5 kernel size
        self.conv2 = nn.Conv2d(32, 64, 5)
        nn.init.xavier_normal_(self.conv2.weight)
        # first fully connected layer
        self.fc1 = nn.Linear(64 * 4 * 4, 256)
        nn.init.xavier_normal_(self.fc1.weight)
        # second fully connected layer
        self.fc2 = nn.Linear(256, 10)
        nn.init.xavier_normal_(self.fc2.weight)
        # dropout
        # self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        # convolution and relu activation, max pooling over 2x2 window
        x = F.max_pool2d(F.torch.tanh(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.torch.tanh(self.conv2(x)), (2, 2))
        # tensors get flattened
        x = x.view(-1, self.num_flat_features(x))
        # using relu activation after fully connected laye
        x = F.torch.tanh(self.fc1(x))
        # last layer
        x = self.fc2(x)
```

c)

Activation	**Train Accuracy**	**Test Accuracy**	**Loss**
Tanh	99.98	91.01	0.8059
Sigmoid	94.12	89.92	298.1224
ELU	97.65	89.33	62.9238

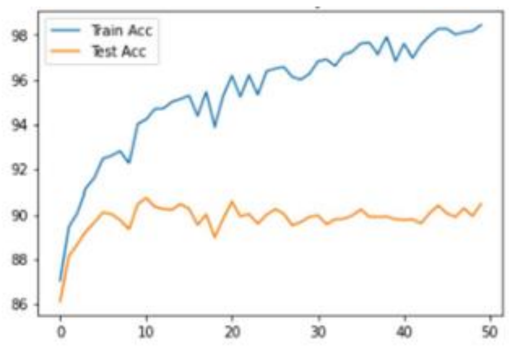
In the table above the results can be observed that Tanh offers the best results in this case, especially when looking at the Loss figure, this is due to its steep activation function. With sigmoid the loss observed with substantially greater as its activation is less steep.

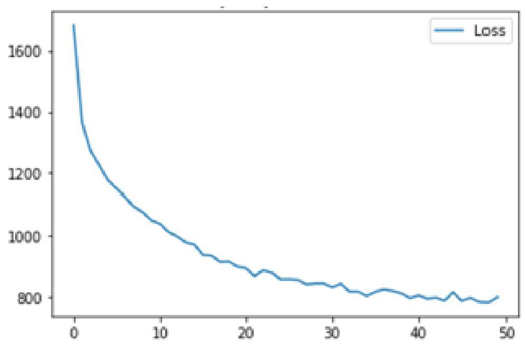
Learning Rate	Train accuracy	Test Accuracy	Loss
0.001	89.02	88.12	602.9023
0.1	97.31	90.43	54.6932
0.5	88.09	83.22	652.7384
1	10	10	4421.9342
10	10	10	4789.3455

When using different learning the results vary greatly. A learning rate that is too small will take too much time to converge and therefore as it can be seen when using 0.001, the loss is substantially large. In order for this learning rate to result in a more satisfactory output the number of epochs would need to be increased too, allowing the loss to converge to a small value. A bigger learning rate, for example 0.1, returns more satisfactory results with an acceptable loss of 54.7. When using a learning rate that is too big the model becomes unreliable, and the loss increases by almost 100 times. It can be deduced that learning rates that are too small result in very slow training while learning rates that are too large result in unstable training.

D)

Dropout rate is a method for regularisation therefore this practice reduces over fitting when training. Dropout freezes the wights during training therefore a higher dropout rate results in a higher change of dropping nodes. When this happens the value of its weight becomes 0. A certain value for the dropout rate also signifies that the training accuracy and learning accuracy will get closer as the dropout rate increase. This allows to control the over fitting of the model however a small dropout rate results in overfitting while a larger one may underfit the results.


Train Acc
Test Acc


Loss