[CM2] Our Own Network

In [ ]:
```
class Simple_CNN(nn.Module):
  def __init__(self, out_1=16, out_2=32, out_3 = 64):
      super(Simple_CNN, self).__init__()
      self.cnn1 = nn.Conv2d(in_channels=1, out_channels=out_1, kernel_siz
      self.relu1 = nn.ReLU()
      self.maxpool1 = nn.MaxPool2d(kernel_size=2,stride=2)
      self.cnn2 = nn.Conv2d(in_channels=out_1, out_channels=out_2, kernel_
      self.relu2 = nn.ReLU()
      self.maxpool2 = nn.MaxPool2d(kernel_size=2,stride=2)
      self.cnn3 = nn.Conv2d(in_channels=out_2, out_channels=out_3, kernel_
      self.maxpool3 = nn.MaxPool2d(kernel_size=2,stride=2)
      self.relu3 = nn.ReLU()
      self.fc1 = nn.Linear(out_2 * 8 * 8, 500)
      self.fc2 = nn.Linear(500, 50)
      self.fc3 = nn.Linear(50,5)
      self.softmax = nn.Softmax()
      self.sig = nn.Sigmoid()

  def forward(self,x):
      out = self.cnn1(x)
      out = self.relu1(out)
      out = self.maxpool1(out)
      out = self.cnn2(out)
      out = self.relu2(out)
      out = self.maxpool2(out)
      out = out.view(out.size(0), -1)
      out = self.fc1(out)
      out = self.relu1(out)
      out = self.fc2(out)
      out = self.relu1(out)
      out = self.fc3(out)
      out = self.softmax(out)
      return out

  def forward2(self,x):
    out = self.cnn1(x)
    out = self.relu1(out)
    out = self.maxpool1(out)
    out = self.cnn2(out)
    out = self.relu2(out)
    out = self.maxpool2(out)
    out = out.view(out.size(0), -1)
    out = self.fc1(out)
    out = self.relu1(out)
    out = self.fc2(out)
    out = self.relu1(out)
    out_enc = out.clone().detach()
    return out_enc
```

```
  def train(self, train_loader,opt, loss_fn, BATCH_SIZE, eval = True):
    loss_tot = 0
    acc = 0
    cnt = 0
    for x,y in zip(train_loader[0], train_loader[1]):
        cnt+=1
        x = x.float()
        x = x.cuda()
        y = y.cuda()
        opt.zero_grad()
        z = self.forward(x)
        y = torch.nn.functional.one_hot(y, num_classes=5)
        loss = loss_fn(z.double(),y.double())
        loss_tot+= loss.item()
        loss.backward()
        opt.step()
    loss_tr = loss_tot/(len(train_loader[1]))
    if eval:
      correct = 0
      for x,y in zip(train_loader[0],train_loader[1]):
        x = x.float()
        x = x.cuda()
        y = y.cuda()
        z = self.forward(x)
        _,yhat = torch.max(z.data,1)
        correct += (yhat== y).sum().item()
      acc = correct/(len(train_loader[1])*BATCH_SIZE)
    return loss_tr,acc

  def eval(self,loader,BATCH_SIZE,loss_fn=nn.CrossEntropyLoss()):
    correct = 0
    acc = 0
    loss_tot = 0
    for x,y in zip(loader[0],loader[1]):
      x = x.float()
      x = x.cuda()
      y = y.cuda()
      z = self.forward(x)
      loss = loss_fn(z.double(),y.long())
      loss_tot += loss.item()
      _ , yhat = torch.max(z.data,1)
      correct += (yhat == y).sum().item()
    acc = correct/(len(loader[1])*BATCH_SIZE)
    loss_val = loss_tot/(len(loader[1])*BATCH_SIZE)
    return loss_val, acc

  def pred(self,x):
    x = x.float()
    x = x.cuda()
    z = self.forward(x)
    _,yhat = torch.max(z.data,1)
    return yhat.item()
```

In [ ]:

```
model2 = Simple_CNN()
model2 = model2.cuda()
summary(model2, (1,32,32))
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 16, 32, 32]             160
              ReLU-2           [-1, 16, 32, 32]               0
         MaxPool2d-3           [-1, 16, 16, 16]               0
            Conv2d-4           [-1, 32, 16, 16]           4,640
              ReLU-5           [-1, 32, 16, 16]               0
         MaxPool2d-6             [-1, 32, 8, 8]               0
            Linear-7                  [-1, 500]       1,024,500
              ReLU-8                  [-1, 500]               0
            Linear-9                   [-1, 50]          25,050
             ReLU-10                   [-1, 50]               0
           Linear-11                    [-1, 5]             255
          Softmax-12                    [-1, 5]               0
================================================================
Total params: 1,054,605
Trainable params: 1,054,605
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.43
Params size (MB): 4.02
Estimated Total Size (MB): 4.46
----------------------------------------------------------------
```

Architecture:
The network architecture summary is shown above. We have 2 convolution layers, each with stride 1 and padding 1. First layer has 16 filters while the second one has 32 filters. The choice of pooling is max pool for all conv layers. The choice of activation function is ReLU. We flatten the output of convolution and employ FCC with 500 followed by a layer containing 50 neurons followed by the output layer. Here too, the choice of activation function is ReLU followed by Softmax at the output layer since our problem is of classification nature.

Optimizer: We have used "adam" optimizer while compiling the model because Adam combines thebest properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm thatcan handle sparse gradients on noisy problems. Adam is relatively easy to confi gure where thedefault confi guration parameters do well on most problems.

Loss Function: We used 'CrossEntropyLoss' since we one hot encoded the output.

In [ ]:
```
opt = Adam(model2.parameters(), lr = 0.001)
crit = nn.CrossEntropyLoss()
model2 = model2.cuda()
tr_loss2 = []
tr_acc2 = []
val_acc2 = []
val_loss2 = []
training_time_tot = 0
for epoch in range(75):
  start_time = time.time()
  l, a = model2.train(train_loader=loader,opt = opt, loss_fn = crit, BATC
  end_time = time.time()
  training_time_tot += end_time - start_time
  print("Epoch ",epoch+1,"--> ",a)
  tr_loss2.append(l)
  tr_acc2.append(a)
  l,a = model2.eval(loader=val_loader,BATCH_SIZE=100)
  val_acc2.append(a)
  val_loss2.append(l)
```

```
Epoch  1 -->  0.7931041666666667
Epoch  2 -->  0.8113541666666667
Epoch  3 -->  0.8221875
Epoch  4 -->  0.8348958333333333
Epoch  5 -->  0.8303958333333333
Epoch  6 -->  0.8438541666666667
Epoch  7 -->  0.8510208333333333
Epoch  8 -->  0.8503541666666666
Epoch  9 -->  0.8568958333333333
Epoch  10 -->  0.8565625
Epoch  11 -->  0.86025
Epoch  12 -->  0.8541458333333334
Epoch  13 -->  0.8589583333333334
Epoch  14 -->  0.86375
Epoch  15 -->  0.8653541666666666
Epoch  16 -->  0.8458333333333333
Epoch  17 -->  0.8689166666666667
Epoch  18 -->  0.8700625
Epoch  19 -->  0.8732708333333333
Epoch  20 -->  0.8670416666666667
Epoch  21 -->  0.87075
Epoch  22 -->  0.8787708333333333
Epoch  23 -->  0.8764791666666667
Epoch  24 -->  0.875875
Epoch  25 -->  0.8773958333333334
Epoch  26 -->  0.8699791666666666
Epoch  27 -->  0.8807291666666667
Epoch  28 -->  0.8785416666666667
Epoch  29 -->  0.8787291666666667
Epoch  30 -->  0.8834375
Epoch  31 -->  0.87875
Epoch  32 -->  0.88175
Epoch  33 -->  0.8771875
Epoch  34 -->  0.8851041666666667
Epoch  35 -->  0.8830625
Epoch  36 -->  0.8865416666666667
```

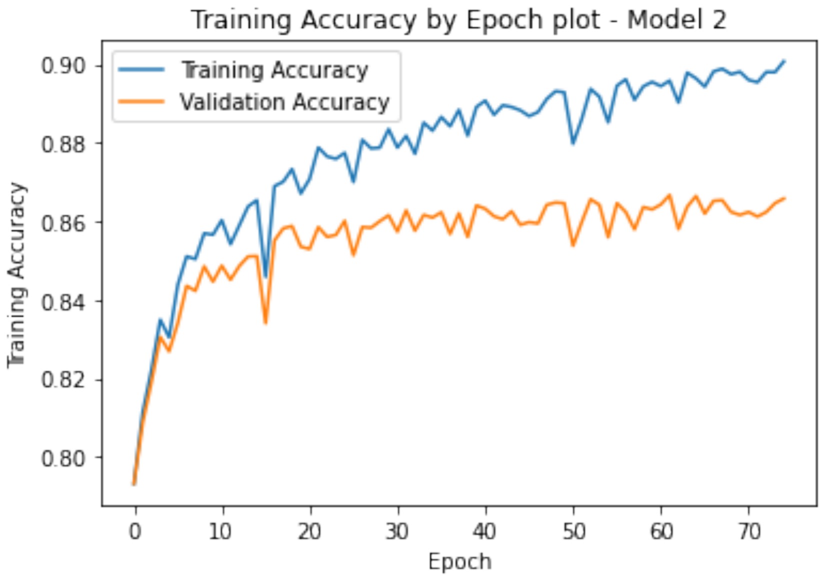
```
Epoch  37 -->  0.8841666666666667
Epoch  38 -->  0.8883541666666667
Epoch  39 -->  0.8817916666666666
Epoch  40 -->  0.8891041666666667
Epoch  41 -->  0.8907083333333333
Epoch  42 -->  0.8871041666666667
Epoch  43 -->  0.8895416666666667
Epoch  44 -->  0.8890833333333333
Epoch  45 -->  0.88825
Epoch  46 -->  0.8867708333333333
Epoch  47 -->  0.8877916666666666
Epoch  48 -->  0.8911041666666667
Epoch  49 -->  0.8931041666666667
Epoch  50 -->  0.8928333333333334
Epoch  51 -->  0.8797291666666667
Epoch  52 -->  0.8860833333333333
Epoch  53 -->  0.8936875
Epoch  54 -->  0.8917083333333333
Epoch  55 -->  0.8851875
Epoch  56 -->  0.8945
Epoch  57 -->  0.8961458333333333
Epoch  58 -->  0.8908958333333333
Epoch  59 -->  0.8943333333333333
Epoch  60 -->  0.8954791666666667
Epoch  61 -->  0.8943958333333333
Epoch  62 -->  0.8958125
Epoch  63 -->  0.8902291666666666
Epoch  64 -->  0.8978333333333334
Epoch  65 -->  0.896375
Epoch  66 -->  0.8942708333333333
Epoch  67 -->  0.8981458333333333
Epoch  68 -->  0.8988125
Epoch  69 -->  0.8974375
Epoch  70 -->  0.8980625
Epoch  71 -->  0.8959791666666667
Epoch  72 -->  0.8953958333333333
Epoch  73 -->  0.898
Epoch  74 -->  0.8979583333333333
Epoch  75 -->  0.9007083333333333
```

In [ ]:

```
plt.plot(tr_acc2, label = 'Training Accuracy')
plt.plot(val_acc2, label = 'Validation Accuracy')
plt.xlabel("Epoch")
plt.ylabel("Training Accuracy")
plt.title("Training Accuracy by Epoch plot - Model 2")
plt.legend()
```

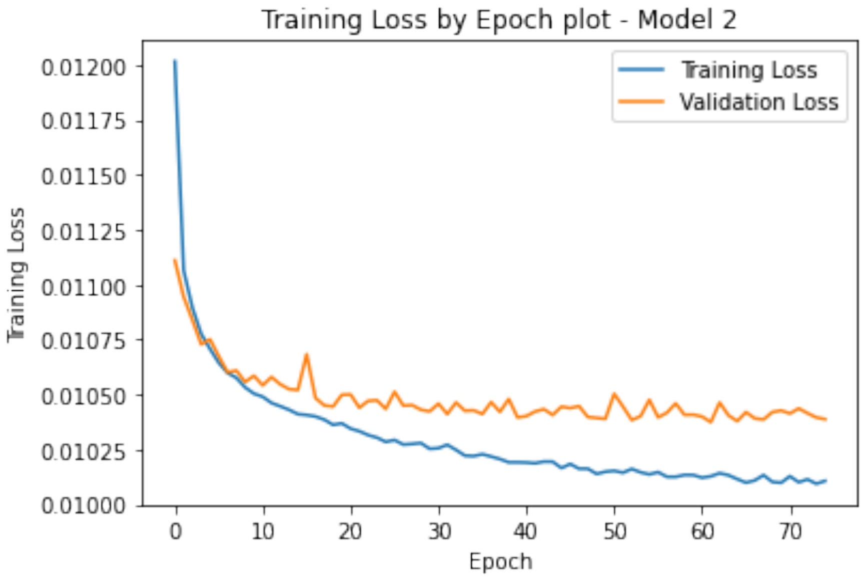
Out[ ]:

```
<matplotlib.legend.Legend at 0x7ff48b4c86d0>
```



In [ ]:
```
plt.plot(np.array(tr_loss2)/100, label = 'Training Loss')
plt.plot(val_loss2, label = 'Validation Loss')
plt.xlabel("Epoch")
plt.ylabel("Training Loss")
plt.title("Training Loss by Epoch plot - Model 2")
plt.legend()
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x7ff48b3c3350>
```



In [ ]:
```
test_loss,test_acc = model2.eval(loader=test_loader,BATCH_SIZE=100)
print("Test loss for Model 2 is ", test_loss)
print("Test accuracy for Model 2 is ", test_acc)
```

```
Test loss for Model 2 is  0.010438133199435765
Test accuracy for Model 2 is  0.86
```

Model 2 - Performance Analysis

Looking at the training and validation set accuracy, we can see that there is no overfitting happening. The test accuracy for this model is 85% which is slightly better than model 1. One constraint that we considered in network design is the complexity of the model since going for model of higher complexity would result in training time increasing a lot which becomes an overhead in performing iterations. Also, complex models are more prone to overfit the data.