CNN vs MLP on MNIST

IFT6135-H2019 Assignment 1

• Akila Daniel Jeeson • Myriam Laiymani • Marc-André Piché • Khalil Slimi

Initialisising Colab drive

4 3 cellules masquées

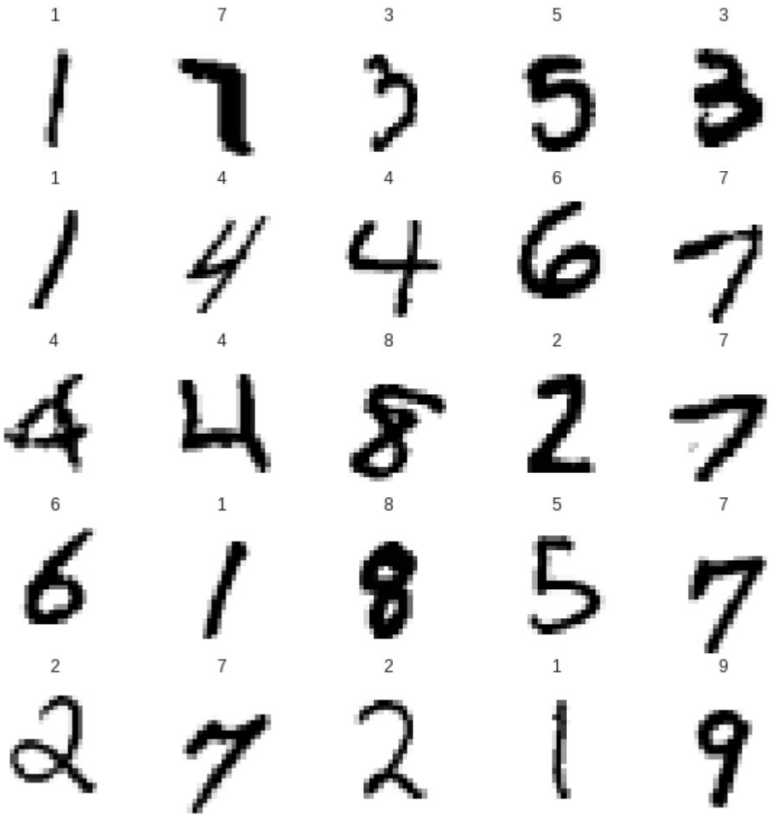
Librairies

4 1 cellules masquées

Dataset

```
[ ]  def showRandSample(dataset, n_row=5, n_col=5):
         nb_img = n_row * n_col

         sample_loader = DataLoader(
           dataset,
           batch_size=nb_img,
           shuffle=True,
           num_workers=1,
           pin_memory=cuda)

         images, labels = next(iter(sample_loader))
         images = images.squeeze()

         fig = plt.figure(figsize=(n_row*2, n_col*2))

         for i in range(1, nb_img +1):
             img = images[i-1]
             ax = fig.add_subplot(n_row, n_col, i)
             ax.set_title(str(labels[i-1].item()))
             plt.axis('off')
             plt.imshow(img,cmap='gray_r')
         plt.show()
```

```
[ ]  showRandSample(train_data)
```



*The best way to view this document is to simply close it and open the source 6135-TP1problem_2.ipynb in Colab

MLP

First as our compairason model we chose one of MLP that reached 97.5 comming out of part 1, which is a simple model with 2 hidden layers of 512 and 768 neurons with Relu as the activation function.

```
[ ]  class MLP_2L(nn.Module):

         def __init__(self):
             super().__init__()

             h1 = 512
             h2 = 768

             # Fully connected layer
             self.fc0 = nn.Linear(28*28, h1)
             self.fc1 = nn.Linear(h1, h2)
             self.fc2 = nn.Linear(h2, 10)

         def forward(self, x):
             x = x.view(-1, 28*28)
             x = F.relu(self.fc0(x))
             x = F.relu(self.fc1(x))
             x = self.fc2(x)

             return x
```

CNN

To expose the effects of convolution on an image classification task, we build a simple CNN with convolution layers replacing a part of the MLP, in this case the first hidden layer.

Simply put : we replace the first hidden layer of the MLP with a convolution block.
So, where we had :

[input]->[hidden 1]->[hidden 2]->[ouput]

we now have :

[input]->[convolution]->[hidden 2]->[ouput]

We tested 3 models CNN1, CNN2, CNN3 with a convolution block of 1 round, 2 rounds and 3 rounds of convolution.

Since our MLP's hidden layers have 512 and 768 neurons respectively and a total of 803k paramaters, we had few choices in designing those layers. For CNN1 it is simply not possible to respect both conditions, we chose to respect the hidden layer; so the total number of parameters is half of the MLP. CNN2 & CNN3 have not only the ~same amount of parameters but also the same hidden layer.

We effectively only **replace** the first hidden layer. It also leads to a nicely balanced number of parameters compared to the MLP. (ie: the convotion part $\approx$ h1 of the MLP). This way, the design patterns stay consistant troughout all of the models.

Here are the designs of the convolution block for the 3 models which all result in a layer of 512 neurons :

CNN1

{[32 , 3×3]->[pooling 7]}

CNN2

{[87 , 3×3]->[pooling 3]} -> {[512 , 3×3]->[pooling 10]}

CNN3

{[64 , 3×3]->[pooling 2]} -> {[77 , 3×3]->[pooling 4]} -> {[512 , 3×3]->[pooling 4]}

87 and 77 might seem like strange numbers, they are chosen simply so that total number of parameters is as close to 803k as possible. The last round of convolution for CNN2&3 must be of 512 filters with pooling down to 1 so that it connects to the hidden layer just like the MLP. The filters are all 3×3 with stride of of 1 so as to remain consistant, the adaptation factor for the size commes only from the padding in the first layer. (by coincidence it is $1, 2, 3$ for CNN 1, 2 ,3)

```
        # Fully connected layer
        self.fc1 = nn.Linear(32*4*4, 768)
        self.fc2 = nn.Linear(768, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 7)

        x = x.view(x.size(0), -1) # flatten

        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x

class CNN_2(nn.Module):

    def __init__(self):
        super().__init__()
        # Conv block
        self.conv1 = nn.Conv2d(1, 87, 3, padding=2)
        self.conv2 = nn.Conv2d(87, 512, 3, padding=1)

        # Fully connected layer
        self.fc1 = nn.Linear(512*1*1, 768)
        self.fc2 = nn.Linear(768, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 3)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 10)

        x = x.view(x.size(0), -1) # flatten

        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x

class CNN_3(nn.Module):

    def __init__(self):
        super().__init__()
        # Conv block
        self.conv1 = nn.Conv2d(1, 64, 3, padding=3)
        self.conv2 = nn.Conv2d(64, 77, 3, padding=1)
        self.conv3 = nn.Conv2d(77, 512, 3, padding=1)

        # Fully connected layer
        self.fc1 = nn.Linear(512*1*1, 768)
        self.fc2 = nn.Linear(768, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.max_pool2d(x, 2)
        x = F.relu(self.conv2(x))
        x = F.max_pool2d(x, 4)
        x = F.relu(self.conv3(x))
        x = F.max_pool2d(x, 4)

        x = x.view(x.size(0), -1) # flatten

        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        return x
```

Model verification and number of parameters :

```
#Number of parameters
from torchsummary import summary

model = CNN_1().to(device)

print("CNN_1 : ",sum(p.numel() for p in model.parameters() if p.requires_grad))
summary(model, (1, 28, 28))


model = CNN_2().to(device)

print("\nCNN_2 : ",sum(p.numel() for p in model.parameters() if p.requires_grad))
summary(model, (1, 28, 28))

model = CNN_3().to(device)

print("\nCNN_3 : ",sum(p.numel() for p in model.parameters() if p.requires_grad))
summary(model, (1, 28, 28))

model = MLP_2L().to(device)

print("\nMLP_2L : ",sum(p.numel() for p in model.parameters() if p.requires_grad))
summary(model, (1, 28, 28))
```

```
CNN_1 :  401994
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 28, 28]             320
            Linear-2                 [-1, 768]         393,984
            Linear-3                  [-1, 10]           7,690
================================================================
Total params: 401,994
Trainable params: 401,994
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.20
Params size (MB): 1.53
Estimated Total Size (MB): 1.73
----------------------------------------------------------------

CNN_2 :  803952
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 87, 30, 30]             870
            Conv2d-2         [-1, 512, 10, 10]         401,408
            Linear-3                 [-1, 768]         393,984
            Linear-4                  [-1, 10]           7,690
================================================================
Total params: 803,952
Trainable params: 803,952
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.99
Params size (MB): 3.07
Estimated Total Size (MB): 4.06
----------------------------------------------------------------

CNN_3 :  802071
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]             640
            Conv2d-2          [-1, 77, 16, 16]          44,429
            Conv2d-3           [-1, 512, 4, 4]         355,328
            Linear-4                 [-1, 768]         393,984
            Linear-5                  [-1, 10]           7,690
================================================================
Total params: 802,071
Trainable params: 802,071
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.72
Params size (MB): 3.06
Estimated Total Size (MB): 3.78
----------------------------------------------------------------

MLP_2L :  803594
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                 [-1, 512]         401,920
            Linear-2                 [-1, 768]         393,984
            Linear-3                  [-1, 10]           7,690
================================================================
Total params: 803,594
Trainable params: 803,594
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 3.07
Estimated Total Size (MB): 3.08
----------------------------------------------------------------
```

```
[ ]  loss_fn = nn.CrossEntropyLoss()
     test_loss_fn = nn.CrossEntropyLoss(reduction='sum')

     #Results directory
     savedir = 'results'
     if not os.path.exists(savedir):
         os.makedirs(savedir)


     ## Saved states are ignored for this expirement

     #checkpoint = torch.load('/content/drive/My Drive/Colab Notebooks//6135/Assignment_1.2')
     #model.load_state_dict(checkpoint['state_dict'])
```

```
[ ]  def test(model, loader, dset = 'validation'):
         model.eval()

         test_loss = 0
         correct = 0
         test_size = 0
         with torch.no_grad():
             for inputs, target in loader:
                 inputs = inputs.to(device,dtype=torch.float)
                 target = target.to(device, dtype=torch.long)

                 output = model(inputs)
                 test_size += len(inputs)
                 test_loss += test_loss_fn(output, target).item()
                 pred = output.max(1, keepdim=True)[1]
                 correct += pred.eq(target.view_as(pred)).sum().item()

         test_loss /= test_size
         accuracy = correct / test_size
         if dset == "validation":
           print('Validation error: {}/{} ({:.2f}%)\n'.format(
               (test_size-correct), test_size,
               100. * (1-accuracy)))
           print('Validation set: Average loss: {:.4f}, Accuracy: {}/{} ({:.2f}%)\n'.format(
               test_loss, correct, test_size,
               100. * accuracy))
         if dset == "train":
           print('\nTraining error: {}/{} ({:.2f}%)'.format(
               (test_size-correct), test_size,
               100. * (1-accuracy)))
         if dset == "test":
           print('\n          Test set Accuracy: {}/{} ({:.2f}%)\n'.format(
               correct, test_size,
               100. * accuracy))



         return test_loss, accuracy
```

```
[ ]  def train(model,loader, optimizer, epoch, scheduler):
         model.train()

         for batch_idx, (inputs, target) in enumerate(loader):
             inputs = inputs.to(device, dtype=torch.float)
             target = target.to(device, dtype=torch.long)

             optimizer.zero_grad()
             output = model(inputs)
             loss = loss_fn(output, target)
             loss.backward()
             optimizer.step()

             if batch_idx % 100 == 0:
                 print(model.__class__.__name__,' Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                     epoch +1, batch_idx * len(inputs), len(loader) *len(inputs) ,
                     100. * batch_idx / len(loader), loss.item()))

         scheduler.step()
         return test(model, loader, dset = 'train')
```

Running the experiment

The experiment being non-determinist, depending on weight initialisation and the draw of the training set, we run the experiment multiple times and take the average results for each correspoding epochs, reseting all weights and sets for each run.

Here : 10 runs of 10 epochs.

```
[ ]  runs = 10
     epochs = 10
```

```
[ ]  import pickle

     def weight_reset(m):
         if isinstance(m, nn.Conv2d) or isinstance(m, nn.Linear):
             m.reset_parameters()


     def MultipleRun(runs,epochs,model):
       liveloss = PlotLosses()
       model = model.to(device)
       name =  model.__class__.__name__

       criterion = nn.CrossEntropyLoss()
       optimizer = optim.SGD(model.parameters(), lr=0.1)
       exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=10, gamma=1.0)


       results = {'name':name, 'loss': [0]*epochs, 'accuracy':[0]*epochs}
       savefile = os.path.join(savedir, results['name']+'.pkl' )

       best_net = '/content/drive/My Drive/Colab Notebooks//6135/Assignment_1.2/best_' + name

       for run in range(runs):
           print("\n","  --  Run : ", run+1)
           since = time.time()
           best_accuracy = 0

           #Reseting all weights for new run
           model.apply(weight_reset)

           log_error = {'train_error':[0]*epochs,'valid_error':[0]*epochs}

           for epoch in range(epochs):
               train_loss, train_acc = train(model, train_loader, optimizer, epoch,exp_lr_scheduler)
               loss, acc = test(model, valid_loader, dset = 'validation')

               if acc > best_accuracy:
                 best_accuracy = acc

                 #Saving best model parameters
                 torch.save({
                     'epoch_based0': epoch,
                     'state_dict': model.state_dict(),
                     'accuracy': acc,
                     'loss' : loss,
                     'optimizer' : optimizer.state_dict(),
                 }, best_net)

                 print('new accuracy parameters saved {}'.format(best_accuracy))

               #Updating average results
               results['loss'][epoch] += loss/runs
               results['accuracy'][epoch] += acc/runs
               log_error['train_error'][epoch] += (1-train_acc)*100
               log_error['valid_error'][epoch] += (1-acc)*100


           _, _ = test(model, test_loader, dset = "test")
           time_elapsed = time.time() - since
           print('Training complete in {:.0f}m {:.0f}s'.format(
                   time_elapsed // 60, time_elapsed % 60))

           ##plotting errors
           fig, ax1 = plt.subplots(figsize=(10, 4))
           ax1.plot(log_error['train_error'], label = "train")
           ax1.plot(log_error['valid_error'], label = "validation")
           ax1.set_ylabel('error')
           ax1.set_xlabel('epochs')
           plt.legend()
           plt.show()

         with open(savefile, 'wb') as fout:
           pickle.dump(results, fout)
```

Running CNN1

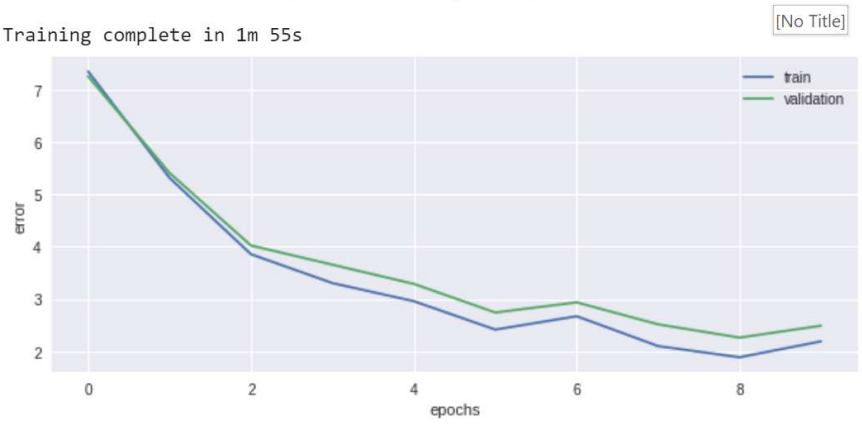
```
[ ]  MultipleRun(runs,epochs,CNN_1())
```

```
      --  Run :  1
CNN_1  Epoch: 1 [0/45056 (0%)]  Loss: 2.294302
CNN_1  Epoch: 1 [12800/45056 (28%)]     Loss: 0.848484
CNN_1  Epoch: 1 [25600/45056 (57%)]     Loss: 0.412616
CNN_1  Epoch: 1 [38400/45056 (85%)]     Loss: 0.264937

Training error: 4098/45000 (9.11%)
Validation error: 1364/15000 (9.09%)

Validation set: Average loss: 0.3017, Accuracy: 13636/15000 (90.91%)

new accuracy parameters saved 0.9090666666666667
CNN_1  Epoch: 2 [0/45056 (0%)]  Loss: 0.218321
CNN_1  Epoch: 2 [12800/45056 (28%)]     Loss: 0.237038
CNN_1  Epoch: 2 [25600/45056 (57%)]     Loss: 0.346456
CNN_1  Epoch: 2 [38400/45056 (85%)]     Loss: 0.156660
```

Running CNN1

```
[ ] MultipleRun(runs,epochs,CNN_1())
```

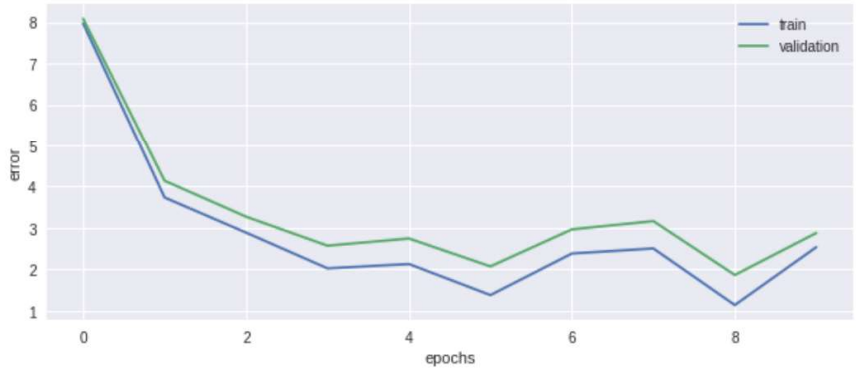
```
CNN_1  Epoch: 7 [38400/45056 (85%)]     Loss: 0.090491

Training error: 1201/45000 (2.67%)
Validation error: 440/15000 (2.93%)

Validation set: Average loss: 0.0914, Accuracy: 14560/15000 (97.07%)

CNN_1  Epoch: 8 [0/45056 (0%)]  Loss: 0.075188
CNN_1  Epoch: 8 [12800/45056 (28%)]     Loss: 0.043178
CNN_1  Epoch: 8 [25600/45056 (57%)]     Loss: 0.061582
CNN_1  Epoch: 8 [38400/45056 (85%)]     Loss: 0.118203

Training error: 944/45000 (2.10%)
Validation error: 377/15000 (2.51%)

Validation set: Average loss: 0.0770, Accuracy: 14623/15000 (97.49%)

new accuracy parameters saved 0.9748666666666667
CNN_1  Epoch: 9 [0/45056 (0%)]  Loss: 0.047588
CNN_1  Epoch: 9 [12800/45056 (28%)]     Loss: 0.039022
CNN_1  Epoch: 9 [25600/45056 (57%)]     Loss: 0.121691
CNN_1  Epoch: 9 [38400/45056 (85%)]     Loss: 0.071599

Training error: 848/45000 (1.88%)
Validation error: 339/15000 (2.26%)

Validation set: Average loss: 0.0718, Accuracy: 14661/15000 (97.74%)

new accuracy parameters saved 0.9774
CNN_1  Epoch: 10 [0/45056 (0%)] Loss: 0.048369
CNN_1  Epoch: 10 [12800/45056 (28%)]    Loss: 0.034491
CNN_1  Epoch: 10 [25600/45056 (57%)]    Loss: 0.045848
CNN_1  Epoch: 10 [38400/45056 (85%)]    Loss: 0.027964

Training error: 985/45000 (2.19%)
Validation error: 373/15000 (2.49%)

Validation set: Average loss: 0.0799, Accuracy: 14627/15000 (97.51%)


          Test set Accuracy: 9775/10000 (97.75%)

Training complete in 1m 55s
```



Running CNN2

```
[ ] MultipleRun(runs,epochs,CNN_2())
```

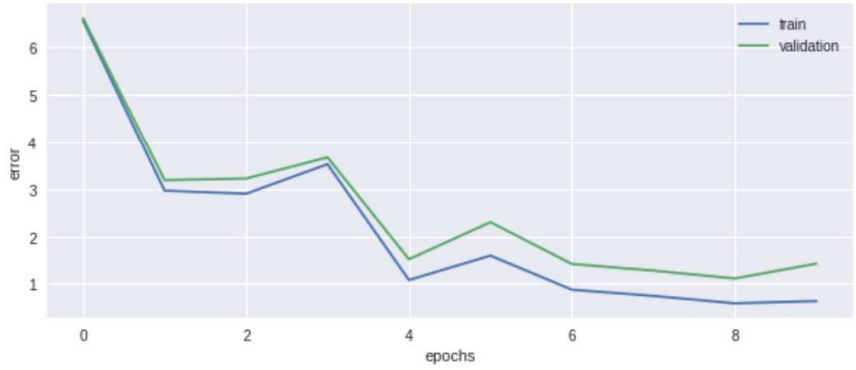
```
CNN_2  Epoch: 7 [38400/45056 (85%)]     Loss: 0.026354

Training error: 1073/45000 (2.38%)
Validation error: 445/15000 (2.97%)

Validation set: Average loss: 0.0936, Accuracy: 14555/15000 (97.03%)

CNN_2  Epoch: 8 [0/45056 (0%)]  Loss: 0.075238
CNN_2  Epoch: 8 [12800/45056 (28%)]     Loss: 0.073145
CNN_2  Epoch: 8 [25600/45056 (57%)]     Loss: 0.040456
CNN_2  Epoch: 8 [38400/45056 (85%)]     Loss: 0.033581

Training error: 1128/45000 (2.51%)
Validation error: 475/15000 (3.17%)

Validation set: Average loss: 0.0953, Accuracy: 14525/15000 (96.83%)

CNN_2  Epoch: 9 [0/45056 (0%)]  Loss: 0.049533
CNN_2  Epoch: 9 [12800/45056 (28%)]     Loss: 0.036218
CNN_2  Epoch: 9 [25600/45056 (57%)]     Loss: 0.029813
CNN_2  Epoch: 9 [38400/45056 (85%)]     Loss: 0.060179

Training error: 514/45000 (1.14%)
Validation error: 280/15000 (1.87%)

Validation set: Average loss: 0.0581, Accuracy: 14720/15000 (98.13%)

new accuracy parameters saved 0.9813333333333333
CNN_2  Epoch: 10 [0/45056 (0%)] Loss: 0.010416
CNN_2  Epoch: 10 [12800/45056 (28%)]    Loss: 0.072094
CNN_2  Epoch: 10 [25600/45056 (57%)]    Loss: 0.033559
CNN_2  Epoch: 10 [38400/45056 (85%)]    Loss: 0.147222

Training error: 1142/45000 (2.54%)
Validation error: 432/15000 (2.88%)

Validation set: Average loss: 0.0999, Accuracy: 14568/15000 (97.12%)



          Test set Accuracy: 9735/10000 (97.35%)

Training complete in 3m 28s
```



Running CNN3

```
[ ]  MultipleRun(runs,epochs,CNN_3())
```

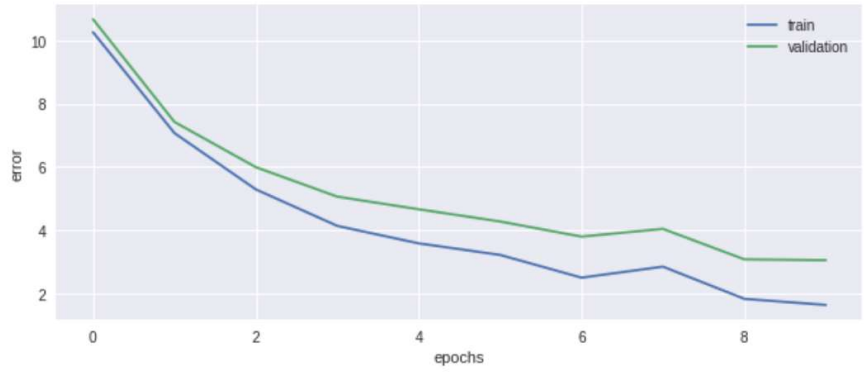
```
Training error: 395/45000 (0.88%)
Validation error: 213/15000 (1.42%)

Validation set: Average loss: 0.0438, Accuracy: 14787/15000 (98.58%)

new accuracy parameters saved 0.9858
CNN_3  Epoch: 8 [0/45056 (0%)]  Loss: 0.044479
CNN_3  Epoch: 8 [12800/45056 (28%)]     Loss: 0.004952
CNN_3  Epoch: 8 [25600/45056 (57%)]     Loss: 0.025091
CNN_3  Epoch: 8 [38400/45056 (85%)]     Loss: 0.019235

Training error: 336/45000 (0.75%)
Validation error: 192/15000 (1.28%)

Validation set: Average loss: 0.0409, Accuracy: 14808/15000 (98.72%)

new accuracy parameters saved 0.9872
CNN_3  Epoch: 9 [0/45056 (0%)]  Loss: 0.013525
CNN_3  Epoch: 9 [12800/45056 (28%)]     Loss: 0.030712
CNN_3  Epoch: 9 [25600/45056 (57%)]     Loss: 0.013097
CNN_3  Epoch: 9 [38400/45056 (85%)]     Loss: 0.009383

Training error: 266/45000 (0.59%)
Validation error: 167/15000 (1.11%)

Validation set: Average loss: 0.0362, Accuracy: 14833/15000 (98.89%)

new accuracy parameters saved 0.9888666666666667
CNN_3  Epoch: 10 [0/45056 (0%)] Loss: 0.007293
CNN_3  Epoch: 10 [12800/45056 (28%)]    Loss: 0.043962
CNN_3  Epoch: 10 [25600/45056 (57%)]    Loss: 0.009454
CNN_3  Epoch: 10 [38400/45056 (85%)]    Loss: 0.009256

Training error: 287/45000 (0.64%)
Validation error: 214/15000 (1.43%)

Validation set: Average loss: 0.0432, Accuracy: 14786/15000 (98.57%)


          Test set Accuracy: 9880/10000 (98.80%)

Training complete in 3m 3s
```



MLP

```
[ ]  MultipleRun(runs,epochs,MLP_2L())
```

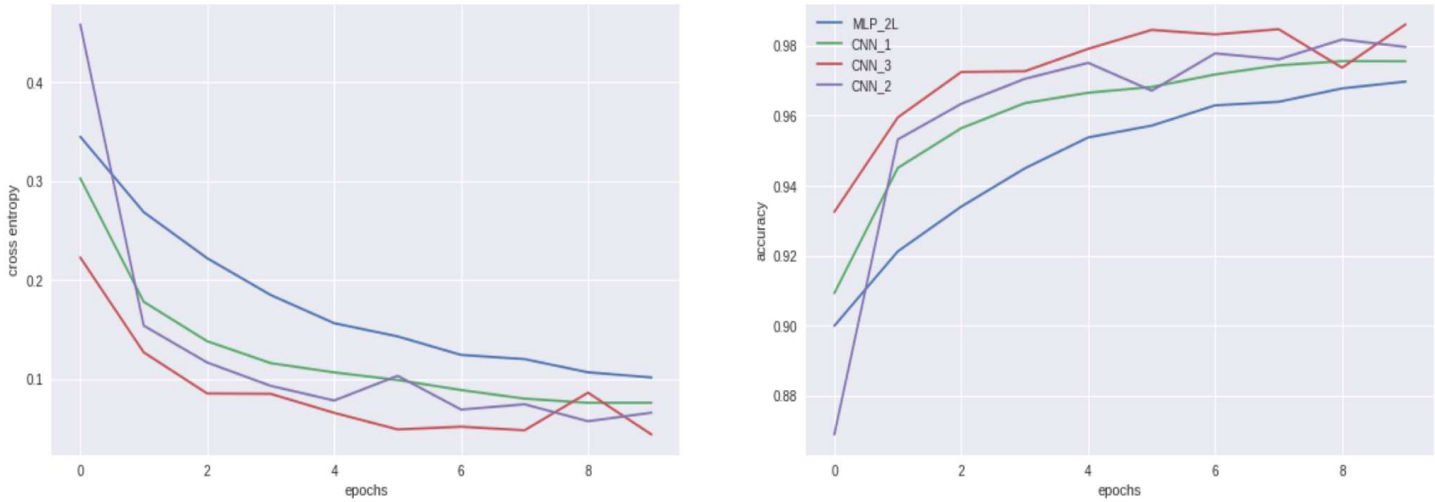
```
Training error: 1119/45000 (2.49%)
Validation error: 568/15000 (3.79%)

Validation set: Average loss: 0.1256, Accuracy: 14432/15000 (96.21%)

new accuracy parameters saved 0.9621333333333333
MLP_2L  Epoch: 8 [0/45056 (0%)] Loss: 0.053017
MLP_2L  Epoch: 8 [12800/45056 (28%)]    Loss: 0.083854
MLP_2L  Epoch: 8 [25600/45056 (57%)]    Loss: 0.218690
MLP_2L  Epoch: 8 [38400/45056 (85%)]    Loss: 0.064988

Training error: 1276/45000 (2.84%)
Validation error: 605/15000 (4.03%)

Validation set: Average loss: 0.1356, Accuracy: 14395/15000 (95.97%)

MLP_2L  Epoch: 9 [0/45056 (0%)] Loss: 0.086680
MLP_2L  Epoch: 9 [12800/45056 (28%)]    Loss: 0.097920
MLP_2L  Epoch: 9 [25600/45056 (57%)]    Loss: 0.077001
MLP_2L  Epoch: 9 [38400/45056 (85%)]    Loss: 0.082225

Training error: 817/45000 (1.82%)
Validation error: 460/15000 (3.07%)

Validation set: Average loss: 0.1025, Accuracy: 14540/15000 (96.93%)

new accuracy parameters saved 0.9693333333333334
MLP_2L  Epoch: 10 [0/45056 (0%)]        Loss: 0.075662
MLP_2L  Epoch: 10 [12800/45056 (28%)]   Loss: 0.106989
MLP_2L  Epoch: 10 [25600/45056 (57%)]   Loss: 0.038961
MLP_2L  Epoch: 10 [38400/45056 (85%)]   Loss: 0.081783

Training error: 730/45000 (1.62%)
Validation error: 456/15000 (3.04%)

Validation set: Average loss: 0.1004, Accuracy: 14544/15000 (96.96%)

new accuracy parameters saved 0.9696

          Test set Accuracy: 9708/10000 (97.08%)

Training complete in 1m 40s
```



Compairing Results

```
[ ]
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

for filename in os.listdir(savedir):
    if filename.endswith('.pkl'):
        with open(os.path.join(savedir, filename),'rb') as fin:
            results = pickle.load(fin)
            ax1.plot(results['loss'], label = filename[:-4])
            ax1.set_ylabel('cross entropy')
            ax1.set_xlabel('epochs')

            ax2.plot(results['accuracy'], label = filename[:-4])
            ax2.set_ylabel('accuracy')
            ax2.set_xlabel('epochs')

plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fe9505a9cc0>
```



Final comments

When looking at the aboves charts, we have to remember that it is plotted from the average of 10 runs of 10 epochs. So what we see is really the evolution of the models during training. For exemple: CNN2 always starts with poor results for the first epoch.

CNN1

As this model slightly outperforms the MLP at every epoch, we can imagine it also doing better in the long run. However, since they end up with the same value on the last epoch, we would need a longer run to confirm that. With half as many parameters as the MLP, it would be tempting to declare this model simply superior to the MLP, but we should not jump to hasty conclusions as we should remember from part 1 that some models with lower number of parameters also performed well, so there might be a redundent amount of neurons that don't contribute much.

Still it is really interesting that we can simply replace 400k parameters with 320 parameters and gain in performance. It clearly shows that a single layers gets more out of the 32 feature maps then a fully connected layer.

CNN2

With a second round of convolution the effect is even more pronounced and the model now clearly outperforms the MLP. However this time we have have a convolution of 512 filters for the missing 400k parameters that make it equivalent to the MLP in number of paramters but much better in performance.

CNN3

Reaching 99% on test accuracy, this model we conserve the same number of paramaters with another round of convolution thanks to more pooling. Compared to CNN2 , mostly only the architecture changed, being deeper and slightly thinner

Not only does CNN3, CNN2 and the MLP have similar number of parameters, but they are also distributed and balanced in a similar way, where the parameters in the first fully connected hidden layer are replaced by the parameters of the convolution.

It is unknown with this experiment how much convolution depth we can add and still gain perfomance, either plateau or loose performance.

What this experiment has shown though with all three models is how much convolution is better then a fully connected layer for image based classification. MNIST being a relatively simple dataset, we suspect that with a more complex task the difference would be even more pronounced and we could really see the difference in capacity.