## 1) MNIST classification

a)

I. A low learning rate can make sure that we do not miss any local minimal points, but it also takes a long time to converge when the value is too small. Inversely, training with a large learning rate would be less time-consuming, but it may lead to overshooting, we will miss the minimum.

II. A low weight decay value would lead to overfitting problem. Inversely, a large weight decay value would lead to underfitting problem.

b)

I. ((28-5+1+2*2)/2-5+1+2*2)/2) = 7, so the dimension of conv2 layer is 7*7*64 = 3136;

II. The input channel of the 2nd Convolution layer should be corresponding to the output channel in 1st Convolution layer. In_channels was corrected to 32.

III. The kernel size of the 2nd Convolution layer should be 5*5

c) I convert fashion mnist module into numpy array and calculated the mean and standard deviation value. Before regularization the accuracy is 82%; And after correction the accuracy rose to 85%.

```
train_data = datasets.FashionMNIST('../data', train=True, download=True,
                transform=transforms.Compose([
                    transforms.ToTensor()
                ]))

test_data = datasets.FashionMNIST('../data', train=False,
                transform=transforms.Compose([
                    transforms.ToTensor()
                ]))
train_dataset_array = train_data.data.numpy()
mean = train_dataset_array.mean() / 255
std = train_dataset_array.std() / 255
print(mean)
print(std)
```

```
[Running] python -u "/Users/zhangbowen
Mean =  0.2860405969887955
Std =  0.3530242445149223
```

Mean = 0.286040; Std = 0.353024

d) Besides adding batch normalization, dropout is used as the regularization method. It is inserted after the first fc layer. Before adding drop_out the accuracy is aroud 84%; After adding drop_out(0.05) normalization and batch_normalization the accuracy rose to 87%.

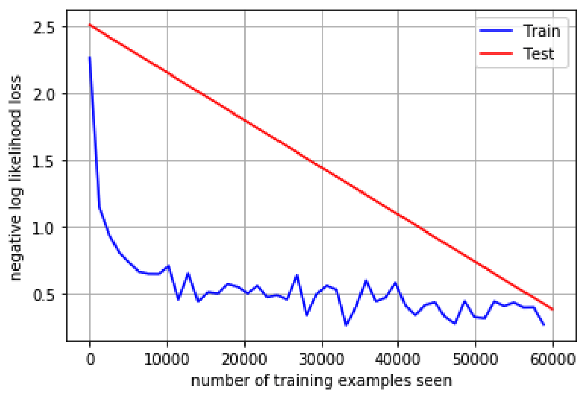
```python
class CNN3(nn.Module):
    def __init__(self):
        super(CNN3, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32,
                               kernel_size=5, stride=1, padding=2)
        self.batch_1 = nn.BatchNorm2d(32)
        self.maxpool = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
                               kernel_size=5, stride=1, padding=2)
        self.batch_2 = nn.BatchNorm2d(64)
        self.fc1 = nn.Linear(in_features=3136, out_features=256)
        self.batch_3 = nn.BatchNorm2d(256)
        self.fc2 = nn.Linear(in_features=256, out_features=10)
        self.batch_3 = nn.BatchNorm1d(256)

        self.drop_out = nn.Dropout(0.05)
        nn.init.kaiming_normal_(self.conv1.weight, nonlinearity='relu')
        nn.init.kaiming_normal_(self.conv2.weight, nonlinearity='relu')
        nn.init.kaiming_normal_(self.fc1.weight, nonlinearity='relu')
        nn.init.kaiming_normal_(self.fc2.weight, nonlinearity='linear')

        # TODO

    def forward(self, x):
        # TODO
        x = self.conv1(x)
        x = self.batch_1(x)
        x = F.relu(x)
        x = self.maxpool(x)
        x = self.conv2(x)
        x = self.batch_2(x)
        x = F.relu(x)
        x = self.maxpool(x)
        x = x.view(-1, 3136)
        x = self.fc1(x)
        x = self.drop_out(x)
        x = self.batch_3(x)
        x = F.relu(x)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

```
Test set: Average loss: 0.3842, Accuracy: 8656/10000 (87%)
```




e)

```
array([[ -42,  131, -170],
       [  42,  169,  114],
       [ -86,   -5,   54]])
```

f) Parameters = 58266; FLOPs = 41636736

## 2) Image Denoising

a) The state-of-art network depth setting method using effective patch sizes as the reference. The relation between receptive field and depth(d) is: receptive field = (2d + 1) * (2d + 1). For the Gaussian denoising, the author decided to refer to EPLL of which the receptive field is 35 * 35. So, the depth is 17.
DnCNN – 3 is designed to learn a single model specific for three general image denoising tasks.

b) Sigma = 25 ; PSNR = 29.26dB, SSIM = 0.9022
Sigma = 45 ; PSNR = 18.37dB, SSIM = 0.4074

c) PSBR = 26.70dB, SSIM = 0.8397

d) PSNR = 16.77dB, SSIM = 0.7096 – pretrained
PSNR = 18.06dB, SSIM = 0.6703 – clean
Select different model by changing the path parameter.

```
parser.add_argument('--model_path', default='models/model_001.pth', type=str, help='the model name')
```

e) Noise data is saved by modifying the return value of forward function. And setting action to 'store_false'.

```
    def forward(self, x):
        y = x
        out = self.dncnn(x)
        noise = out.data.cpu().numpy()
        np.save('noise', noise)
        return out
```

```
parser.add_argument('--save_result', action='store_false', help='save the denoised image') # set action to false
```

The value of Gaussian noise fetched from the picture range from [-0.633, 0.650]. I firstly amplify the range to [-255, 255], then added 127 to the array. Finally set those value less than 0 to 0 and greater than 255 to 255.
The standard deviation of the noise: std = 52.60797

```
noise_array = noise_array / noise_array.max() *255
noise_array = noise_array + 127

ind_neg = noise_array < 0
ind_plu = noise_array > 255

noise_array[ind_neg] = 0
noise_array[ind_plu] = 255
```

## 3) Semantic segmentation

a)
   I. Simple upsampling can be realized by bilinear interpolation. In FCN model the upsampling chose backward convolution to match the original image size.
   II. Some skips which combine the final prediction layer with lower layers with finer strides are added. Because the shallower layers contain more detail of edge.

b) My given image contains the class of person (classes = 15) and sheep (classes = 17). I noticed that using the given *fnc* model, it will fill the area with the corresponding class value of person and sheep. So, I set all the class value of 15 (representing person) to zero, so that the segmentation picture only contains the value of sheep class.

```
# calculate labels
om = torch.argmax(out.squeeze(), dim=0).detach().cpu().numpy() #om size 480 * 640
print (np.unique(om)) #输出矩阵中unique value

person_index = om == 15 #15 represents person, set those area to zero, so that it wo
om[person_index] = 0
```

Then change the color of sheep from (128, 64, 0) to (255,0,0) and apply it to the original picture.

c) I read the given mask picture and separate it into r, g, b channels. Because the backgroud color is (0, 0, 0). So I only need to convert the color of person class to (0, 0, 0) so that only the needed sheep class showing on the mask picture.

```
mask_pic = Image.open('./BZHANG@TCD.IE_mask.png')
sheep_seg = Image.open('./sheep_segmentation.png')

mask_pic = np.array(mask_pic)
#remove people in mask pic
r_channel, g_channel, b_channel = cv2.split(mask_pic)
ind_r = r_channel == 192
ind_g = g_channel == 128
ind_b = b_channel == 128

r_channel[ind_r] = 0
g_channel[ind_g] = 0
b_channel[ind_b] = 0

#array reomoved people class = 15
mask_array = cv2.merge(r_channel, g_channel, b_channel)

mask_pic = Image.fromarray(mask_array, 'RGB')
```

Then I calculated the number of pixels for overlapped and union part.

IOU rate = 0.79265

**Figure 1. Sheep separated from given mask**

**Figure 2. Sheep separated from model**