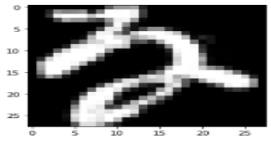
Part 0

```
In [10]: ### 1000: visualize a sample image and corresponding label from KMNIST
         ldr = iter(kmnist_train_loader)
         ims, lbls = ldr.next()
         plt.imshow(ims[0].numpy().squeeze(), cmap='gray')
         print(lbls [0].numpy())
```

```
4
```



Part 1:

Activation function: Sigmoid, Relu and identity

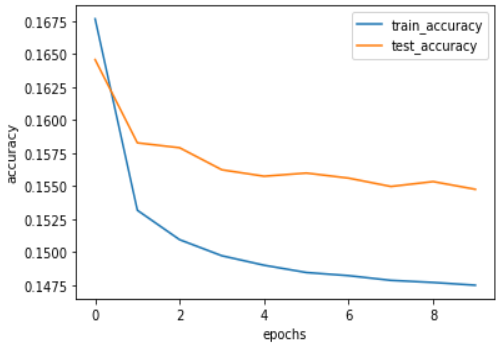
Cross Entropy Function: Softmax and cross entropy functions were implemented.

Network and training

After training, the results were alike

```
Epoch: 0
        training loss/accuracy: 0.180/0.616
        testing loss/accuracy: 0.176/0.672
Epoch: 1
        training loss/accuracy: 0.164/0.803
        testing loss/accuracy: 0.172/0.719
Epoch: 2
        training loss/accuracy: 0.161/0.827
        testing loss/accuracy: 0.170/0.731
Epoch: 3
        training loss/accuracy: 0.160/0.838
        testing loss/accuracy: 0.169/0.755
Epoch: 4
        training loss/accuracy: 0.160/0.847
        testing loss/accuracy: 0.168/0.767
Epoch: 5
        training loss/accuracy: 0.159/0.850
        testing loss/accuracy: 0.168/0.760
Epoch: 6
        training loss/accuracy: 0.159/0.855
        testing loss/accuracy: 0.168/0.777
Epoch: 7
        training loss/accuracy: 0.159/0.860
        testing loss/accuracy: 0.168/0.767
Epoch: 8
        training loss/accuracy: 0.158/0.874
        testing loss/accuracy: 0.163/0.821
Epoch: 9
        training loss/accuracy: 0.156/0.886
        testing loss/accuracy: 0.163/0.817
```

Figure 0: Training of feed forward



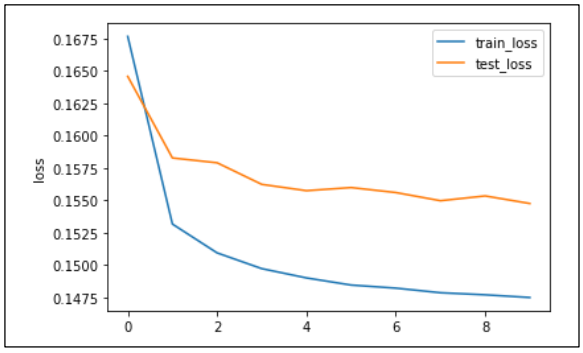


Figure 1: Graph for training and testing accuracy Figure 2: Graph for training and testing loss

Part 2:

Convolution Neural Network

In model of conv_net,

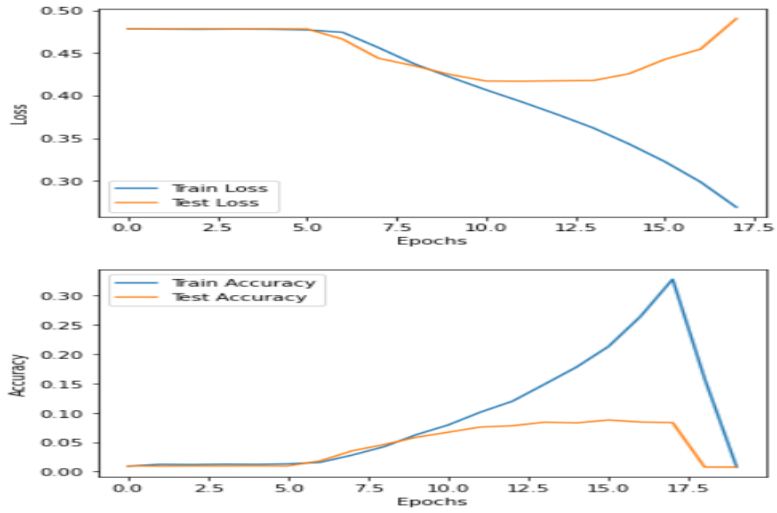


Figure 4: Top image is about training and testing loss, lower image is for training and testing accuracy.

Part 3:

My conv2D code runs but it is too time consuming. So, could not generate graph due to the shortage of time.

```
### TODO: Set target device for computations
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

### TODO: Initialize my_conv2d
model = my_conv2d(in_channels=3, out_channels=1, kernel_size=3)
### TODO: Put model parameters on target device
model.to(device)

train_loss_tracker = []
train_accuracy_tracker = []

test_loss_tracker = []
test_accuracy_tracker = []

for epoch in range(epochs):
    print(f'Epoch: {epoch}')
    train_loss, train_accuracy = Train(model, train_dl, device)
    test_loss, test_accuracy = Test(model, test_dl, device)
    train_loss_tracker.append(sum(train_loss)/len(train_loss))
    train_accuracy_tracker.append(sum(train_accuracy)/len((train_accuracy)))
    test_loss_tracker.append(test_loss)
    test_accuracy_tracker.append(test_accuracy)
    print('\t training loss/accuracy: {0:.3f}/{1:.3f}'.format(sum(train_loss)/len(train_loss), sum(train_accuracy)/len((train_ac
    print('\t testing loss/accuracy: {0:.3f}/{1:.3f}'.format(test_loss, test_accuracy))

torch.save(model.state_dict(), 'model_extra_credit.pth')
```

```
Epoch: 0
```

```
class my_conv2d(nn.Module):
    ### EXTRA CREDIT FOR UNDERGRADUATES - ### TODO: Compulsory for graduates
    ### Complete this class to have the perform the convolution similar to torch.nn.Conv2d
    def __init__(self, in_channels, out_channels, kernel_size, stride=1):
        super(my_conv2d, self).__init__()
        self.in_channels = in_channels
        self.out_channels = out_channels
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = 0
        self.dialation = 1
        self.groups = 1
        self.k = np.sqrt(self.groups / self.in_channels * self.kernel_size**2)
        self.bias = torch.FloatTensor(self.out_channels).uniform_(-self.k, self.k)
        self.weight = torch.FloatTensor(self.out_channels, self.in_channels//self.groups, self.kernel_size, self.kernel_size).uni

    def forward(self, x):
        w_out = (x.size(2) - self.weight.size(2))//self.stride + 1
        h_out = (x.size(3) - self.weight.size(3))//self.stride + 1
        x_unfold = torch.nn.functional.unfold(x, kernel_size=(self.kernel_size, self.kernel_size))
        output_unfolded = x_unfold.transpose(1, 2).matmul(self.weight.view(self.weight.size(0), -1).t()).transpose(1, 2)
        output_folded = torch.nn.functional.fold(output_unfolded, output_size=(w_out, h_out), kernel_size=(1,1))
        return output_folded
```

Figure: training and Myconv2D implementation

Ablation of Study

```
### TODO: visualize the sample image
plt.imshow(sample_image.permute([1,2,0]))
plt.show()
```

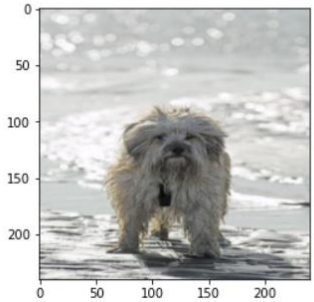


Figure 5: Sample image for AlexNet

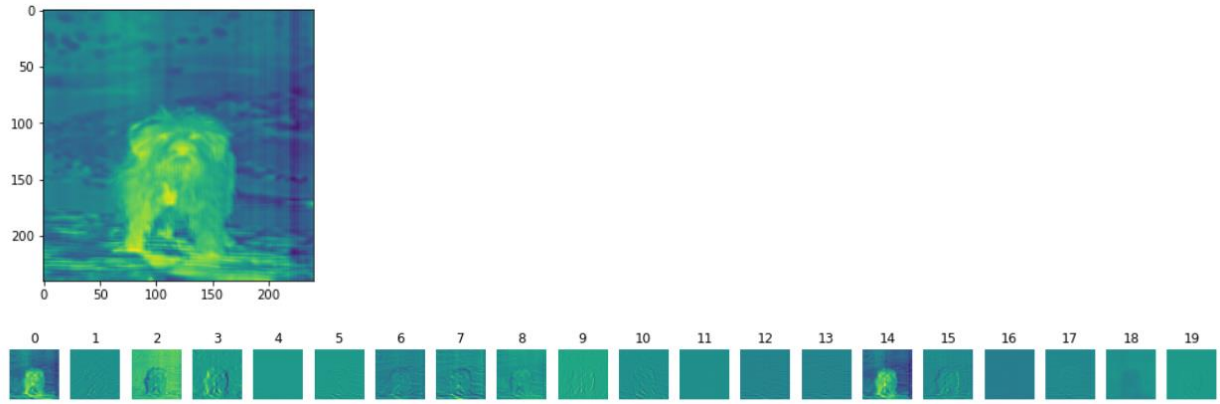


Figure 6: 20 images after convolution