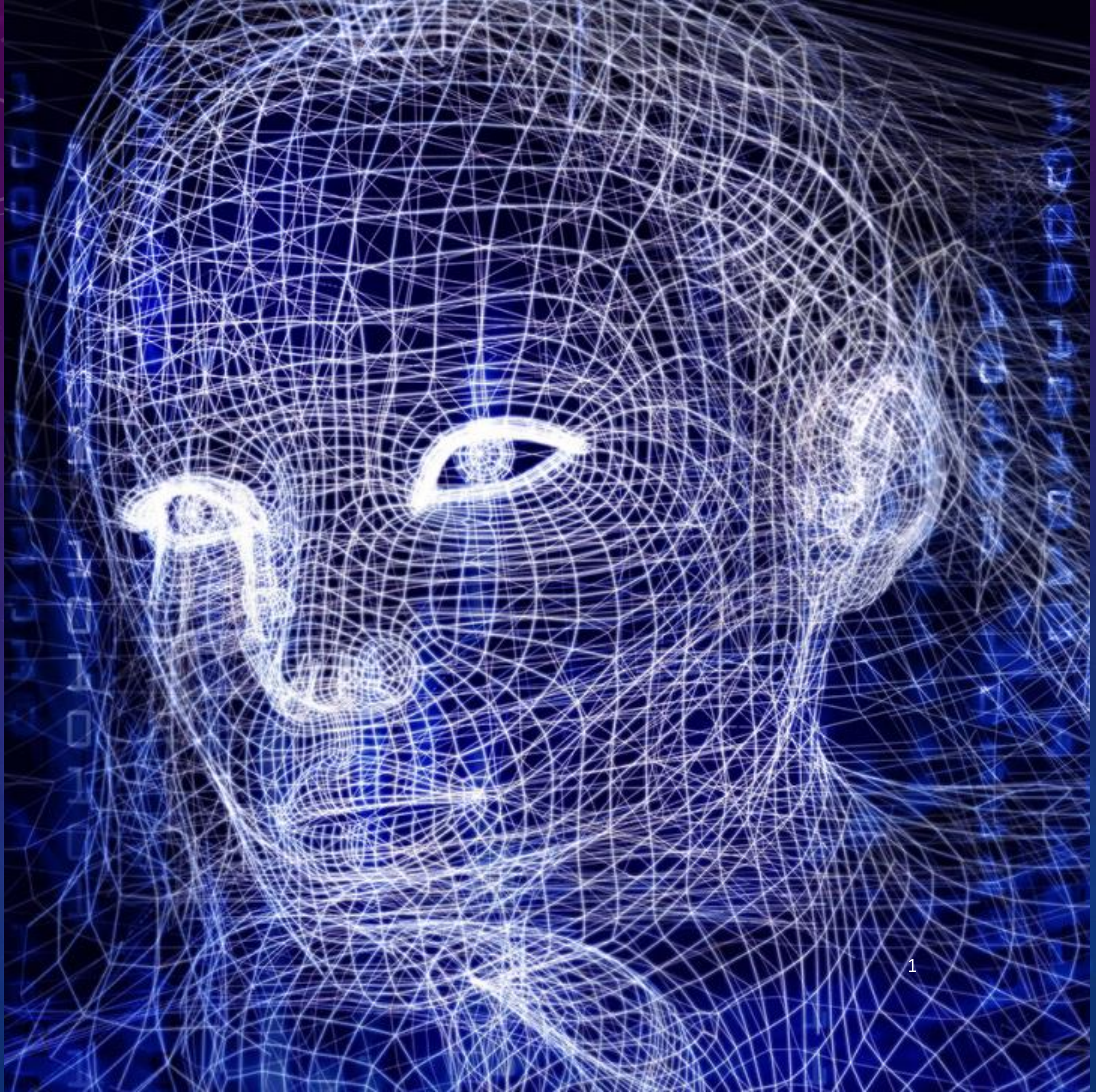
# COMPUTER VISION AND ITS APPLICATIONS

# CH 3_EXERCISE

徐繼聖
Gee-Sern Jison Hsu
National Taiwan University of Science and Technology



# Example 3-1: Convolution and Deconvolution

- Please download the "3-1_Convolution and Deconvolution.zip" from the Moodle.
- Upload the "sample.jpg" to the Google Colab.
- Run the codes and get the output images through the convolution and deconvolution.

# Example 3-1: Convolution and Deconvolution

```
# First read the input image
img = cv2.imread('./sample.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(112,112))
plt.imshow(img)
img = torch.from_numpy(img.transpose((2, 0, 1)))
img_tensor = img.float().div(255).unsqueeze(0)
print('*****Dimensions of input image****** ',img_tensor.shape)
```

Image path

Resize the image to 112 by 112

We need to transpose the image in order to match the format of pytorch
The format of image:
Numpy: ( Height, Width, Channel)
Torch.Tensor( Channel, Height, Width)

# Example 3-1: Convolution and Deconvolution

Create the Network with 1 forward convolution

```
In [4]:  #Define the neural Network
         class Convolution(nn.Module):
           def __init__(self):
             super(Convolution, self).__init__()
             kernel = [[1/9, 1/9, 1/9],
                       [1/9, 1/9, 1/9],
                       [1/9, 1/9, 1/9]]
             kernel = torch.FloatTensor(kernel).unsqueeze(0).unsqueeze(0)
             self.weight = nn.Parameter(data=kernel, requires_grad=False)

           def forward(self, x):
             x1 = x[:, 0]
             x1 = F.conv2d(x1.unsqueeze(1), self.weight, stride=2, padding=0)

             return x1
```

Filter parameters

Single channel and create the forward path.

# Example 3-1: Convolution and Deconvolution

Feed the input image to the convolution

```
5]:  #create an instance of our FilterClass
Convolution1 = Convolution()

print('shape',img_tensor.shape)
out = Convolution1(img_tensor)
#out1=out.copy()
print(out.size())
#print(out)

img_out = out.mul(255).byte()
img_out = img_out.cpu().numpy().squeeze(0).transpose((1, 2, 0))
print('*****Dimensions of output image******:',img_out.shape)

plt.imshow(img_out[:,:,0])
```

Show the image.

# Example 3-1: Convolution and Deconvolution

Create the Deconvolution Network with 1 deconvolution layer

```python
#Define the class Deconvolution
class Deconvolution(nn.Module):
    def __init__(self):
        super(Deconvolution, self).__init__()

        kernel = [[1/9, 1/9, 1/9],
                  [1/9, 1/9, 1/9],
                  [1/9, 1/9, 1/9]]
        #kernel3d =[kernel,kernel,kernel]

        kernel = torch.FloatTensor(kernel).unsqueeze(0).unsqueeze(0)
        self.weight = nn.Parameter(data=kernel, requires_grad=False)

    def forward(self, x):
        x1 = x[:, 0]
        x1 = F.conv_transpose2d(x1.unsqueeze(1), self.weight, stride=2, padding=0)
        return x1
```

# Example 3-1: Convolution and Deconvolution

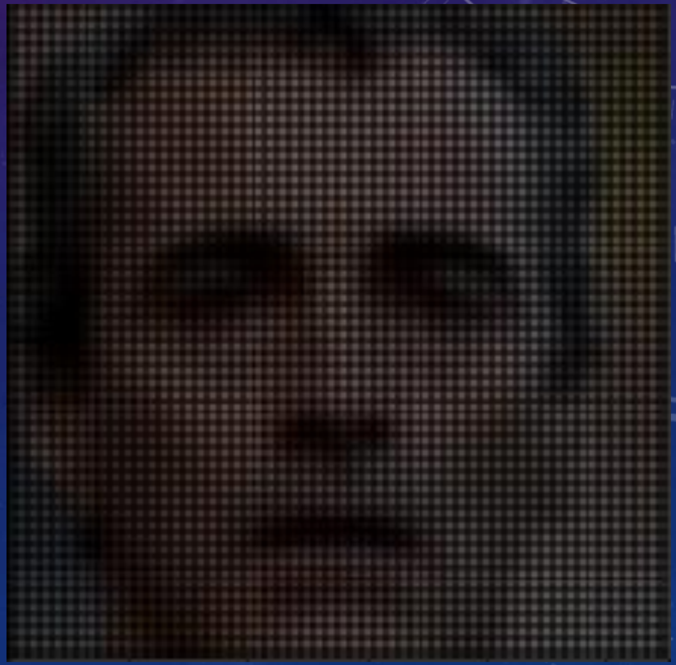
Perform the deconvolution

```
#create an instance of our Deconvolution Class


Deconvolution1 = Deconvolution()
out = Deconvolution1(out)
print(out.size())
#print(out)

img_out = out.mul(255).byte()
img_out = img_out.cpu().numpy().squeeze(0).transpose((1, 2, 0))
print('*****Dimensions of output image******:',img_out.shape)

print(plt.imshow(img_out[:,:,0]))
```



# Exercise 3-1: Convolution and Deconvolution

- Please download the “3-1_Convolution and Deconvolution.zip” from the Moodle.
- Upload the “sample.jpg” to the Google Colab.
- Follow the example code and design a convolution kernal and a deconvolution kernal.
- Run the codes and get the output images through the designed convolution and deconvolution kernals.

Please write down your results and codes in MS Word, then upload to the Moodle.

# Example 3.2 : Autoencoder

- Please download the “exercise3.2_Autoencoder.ipynb” on Moodle.
  - Train the autoencoder and compare the images that reconstruct from the different epoch.
  - Change the encoder and decoder to the below architecture and compare the difference.

Please copy your results and code and paste to a MS Word, then upload to Moodle.

# Example 3.2 : Autoencoder

Define the hyper-parameter and load the training data

```
num_epochs = 10
batch_size = 32
learning_rate = 1e-3
```

Define the hyperparameter

```
img_transform = transforms.Compose([transforms.ToTensor()])
```

Download the Mnist dataset to the folder './data'

```
dataset = torchvision.datasets.FashionMNIST(root='./data', train=True, download=True,
transform=img_transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, shuffle=False)
```

# Example 3.2 : Autoencoder

Define the model architecture

```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True),
            nn.Linear(64, 12),
            nn.ReLU(True),
            nn.Linear(12, 3))
        self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(True),
            nn.Linear(12, 64),
            nn.ReLU(True),
            nn.Linear(64, 128),
            nn.ReLU(True), nn.Linear(128, 28 * 28), nn.Tanh())
```

```
def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

```
autoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=64, out_features=12, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=12, out_features=3, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=3, out_features=12, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=12, out_features=64, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=64, out_features=128, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=128, out_features=784, bias=True)
    (7): Tanh()
  )
)
```

# Example 3.2 : Autoencoder

Define the model architecture

```
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(28 * 28, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True),
            nn.Linear(64, 12),
            nn.ReLU(True),
            nn.Linear(12, 3))
```

Define the encoder

Define the decoder

```
self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(True),
            nn.Linear(12, 64),
            nn.ReLU(True),
            nn.Linear(64, 128),
            nn.ReLU(True),
            nn.Linear(128, 28 * 28),
nn.Tanh())

def forward(self, x):
        x = self.encoder(x)
        x = self.decoder(x)
        return x
```

# Example 3.2 : Autoencoder

Define the model architecture

```
autoencoder(
  (encoder): Sequential(
    (0): Linear(in_features=784, out_features=128, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=64, out_features=12, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=12, out_features=3, bias=True)
  )
  (decoder): Sequential(
    (0): Linear(in_features=3, out_features=12, bias=True)
    (1): ReLU(inplace=True)
    (2): Linear(in_features=12, out_features=64, bias=True)
    (3): ReLU(inplace=True)
    (4): Linear(in_features=64, out_features=128, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=128, out_features=784, bias=True)
    (7): Tanh()
  )
)
```

# Example 3.2 : Autoencoder

Use the Mean Square Error as the loss function

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=1e-5)
```

```
for epoch in range(num_epochs):
    for data in dataloader:
        img, _ = data
        img = img.view(img.size(0), -1)
        img = Variable(img).cuda()
        output = model(img)
        loss = criterion(output, img)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    print('epoch [{}/{}], loss:{:.4f}'.format(epoch + 1, num_epochs, loss.item()))
```

Flatten the input images

Compute  the loss

# Example 3.2 : Autoencoder

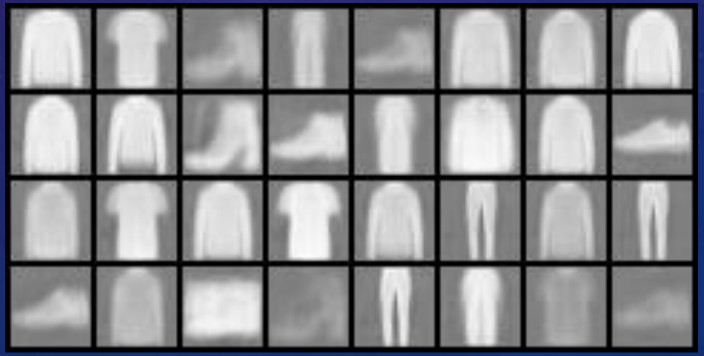
```
if epoch % 1 == 0:
    img = to_img(img.cpu().data)
    save_image(img, ‘./AE_img/input_{}.png'.format(epoch))
    pic = to_img(output.cpu().data)
    save_image(pic, './ AE _img/output_{}.png'.format(epoch))
```

Save the input images

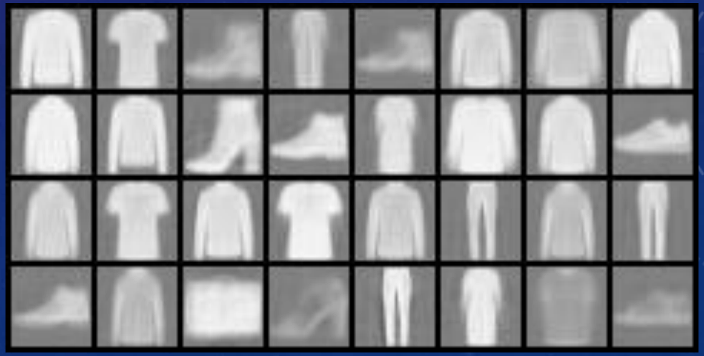
Save the reconstruction images

Input images

Epoch 1 : reconstruction images



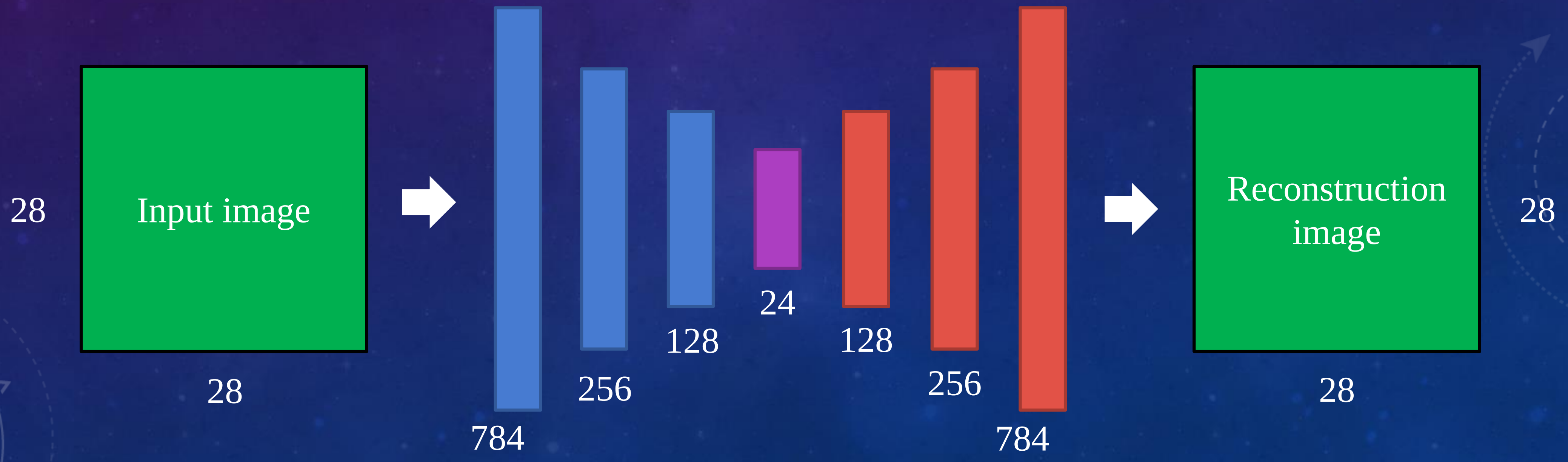
Epoch 10 : reconstruction images



# Exercise 3.2 : Autoencoder

- Please download the “exercise3.2_Autoencoder.ipynb” on Moodle.
    - Train the autoencoder and compare the images that reconstruct from the different epoch.
    - Change the encoder and decoder to the below architecture and compare the difference.

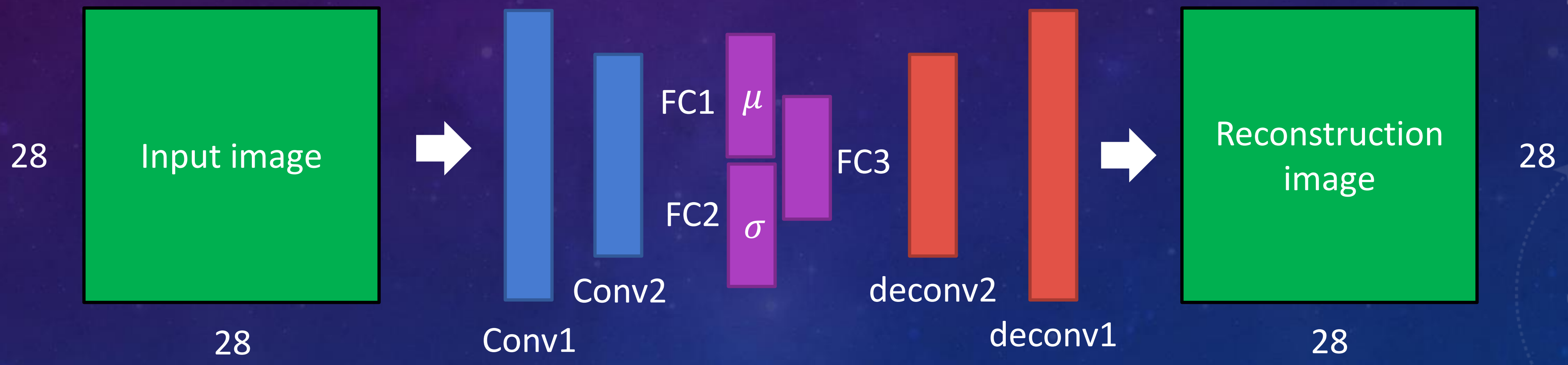
Please copy your results and code and paste to a MS Word, then upload to Moodle.



# **Example 3.3 :** VAE

- Please download the “exercise4.3_VAE.ipynb” on Moodle.

  Train the autoencoder and compare the images that reconstruct from the different epoch.



# Example 3.3 : VAE

- Here we have a CNN network with 2 convolution layers using ReLU follow by one fully connected layer to generate 20 μ and another fully connected layer for 20 σ.

```
self.conv1 = nn.Conv2d(image_channels, 32, kernel_size=4, stride=2)
self.conv2 = nn.Conv2d(32, 64, kernel_size=4, stride=2)

self.fc1 = nn.Linear(h_dim, z_dim)
self.fc2 = nn.Linear(h_dim, z_dim)  Latent vector dimension

def encode(self, x):
    h_1 = F.relu(self.conv1(x))
    h_2 = F.relu(self.conv2(h_1))
    h_3 = F.relu(self.conv3(h_2))
    flat = h_3.view(h_3.size(0), -1)
    z, mu, logvar = self.bottleneck(flat)
    return z, mu, logvar
def bottleneck(self, h):
    mu, logvar = self.fc1(h), self.fc2(h)
    z = self.reparameterize(mu, logvar)
    return z, mu, logvar
```

# Example 3.3 : VAE

- The decoder feeds the 20 latent variables to a fully connected layer followed with 2 transpose convolution layer with ReLU. The output is then feed into a sigmoid layer to generate the image.

```
self.fc3 = nn.Linear(z_dim, h_dim)

self.deconv2 = nn.ConvTranspose2d(64, 32, kernel_size=5, stride=2)
self.deconv1 = nn.ConvTranspose2d(32, image_channels, kernel_size=4, stride=2)


def decode(self, z):
    fc = self.fc3(z)
    reshape = fc.view(fc.size(0), 64, 5, 5)
    h_1 = F.relu(self.deconv1(reshape))
    h_2 = F.relu(self.deconv2(h_1))
    h_3 = self.deconv3(h_2)
    return F.sigmoid(h_3)
```

# **Example 3.3 :** VAE

- We use the encoder to encode the input image. Use sampling to generate z from the mean and variance of the gaussian distribution and then decode it.

```
def reparameterize(self, mu, logvar):
    std = logvar.mul(0.5).exp_()
    if torch.cuda.is_available():
        eps = torch.cuda.FloatTensor(std.size()).normal_()
    else:
        eps = torch.FloatTensor(std.size()).normal_()
    eps = Variable(eps)
    z = mu + std * eps
    return z

def bottleneck(self, h):
    mu, logvar = self.fc1(h), self.fc2(h)
    z = self.reparameterize(mu, logvar)
    return z, mu, logvar
```

# **Example 3.3 :** VAE

- The forward method is defined how does the sequence of data between the layers.

```
def forward(self, x):
    z, mu, logvar = self.encode(x)
    z = self.decode(z)
    return z, mu, logvar
```

# Example 3.3 : VAE

- We define a generation loss which measures the difference between the original and the decoded message using the mean square error. The latent loss measures the difference between gaussian function of the image from a normal distribution using KL-Divergence.

```
def loss_function(recon_x, x, mu, logvar):
    """
    recon_x: generating images
    x: origin images
    mu: latent mean
    logvar: latent log variance
    """
    BCE = reconstruction_function(recon_x, x)  # mse loss
    # loss = 0.5 * sum(1 + log(sigma^2) - mu^2 - sigma^2)
    KLD_element = mu.pow(2).add_(logvar.exp()).mul_(-1).add_(1).add_(logvar)
    KLD = torch.sum(KLD_element).mul_(-0.5)
    # KL divergence
    return BCE + KLD
```

- We use the “Adam” optimizer to train both networks.

```
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
```

# Example 3.3 : VAE

```
for epoch in range(num_epochs):
    model.train()
    train_loss = 0
    for batch_idx, data in enumerate(dataloader):
        img, _ = data
        img = Variable(img)
        if torch.cuda.is_available():
            img = img.cuda()
        optimizer.zero_grad()
        recon_batch, mu, logvar = model(img)
        loss = loss_function(recon_batch, img, mu, logvar)
        loss.backward()
        train_loss += loss.item()
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch,
                batch_idx * len(img),
                len(dataloader.dataset), 100. * batch_idx / len(dataloader),
                loss.item() / len(img)))

    print('====> Epoch: {} Average loss: {:.4f}'.format(
        epoch, train_loss / len(dataloader.dataset)))
    if epoch % 1 == 0:
        save = img.cpu().data
        save_image(img, './vae_img/input_{}.png'.format(epoch))
        save = recon_batch.cpu().data
        save_image(save, './vae_img/image_{}.png'.format(epoch))

torch.save(model.state_dict(), './vae.pth')
```

Send the img to Model

Calculate the loss for backpropagation

# **Example 3.3 :** VAE

```
if epoch % 1 == 0:
    save = img.cpu().data
    save_image(img, './vae_img/input_{}.png'.format(epoch))
    save = recon_batch.cpu().data
    save_image(save, './vae_img/output_{}.png'.format(epoch))
```

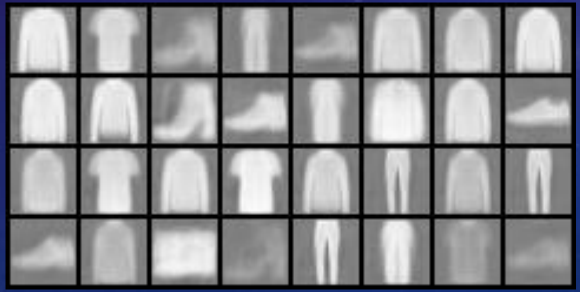
Save the input images

Save the reconstruction images

Epoch: 0

Input

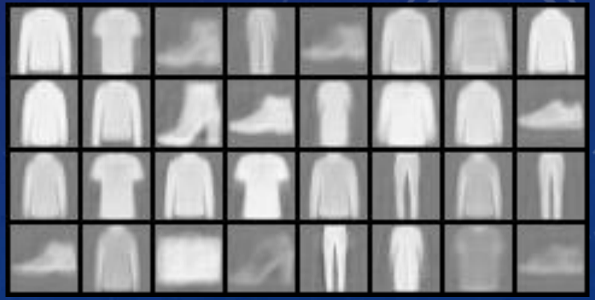
Reconstruct



Epoch: 4

Input

Reconstruct



# Exercise 3.3 - VAE

- Please download the “exercise3.3_VAE.ipynb” on Moodle.
- Please use different latent vector dimension
    1. 1 dimension
    2. 10 dimension
    3. 100 dimension

Please copy your results and code and paste to a MS Word, then upload to Moodle.