In [ ]:

```
import tensorflow.keras.layers
import tensorflow.keras.models
import tensorflow.keras.optimizers
import tensorflow.keras.datasets
import numpy
import matplotlib.pyplot as plt
```

Dense Autoencoder

In [ ]:

```
# Encoder
x = tensorflow.keras.layers.Input(shape=(784), name="encoder_input")

encoder_dense_layer1 = tensorflow.keras.layers.Dense(units=300, name="encoder_dense_1")
encoder_activ_layer1 = tensorflow.keras.layers.LeakyReLU(name="encoder_leakyrelu_1")(en

encoder_dense_layer2 = tensorflow.keras.layers.Dense(units=2, name="encoder_dense_2")(e
encoder_output = tensorflow.keras.layers.LeakyReLU(name="encoder_output")(encoder_dense

encoder = tensorflow.keras.models.Model(x, encoder_output, name="encoder_model")
encoder.summary()

# Decoder
decoder_input = tensorflow.keras.layers.Input(shape=(2), name="decoder_input")

decoder_dense_layer1 = tensorflow.keras.layers.Dense(units=300, name="decoder_dense_1")
decoder_activ_layer1 = tensorflow.keras.layers.LeakyReLU(name="decoder_leakyrelu_1")(de

decoder_dense_layer2 = tensorflow.keras.layers.Dense(units=784, name="decoder_dense_2")
decoder_output = tensorflow.keras.layers.LeakyReLU(name="decoder_output")(decoder_dense

decoder = tensorflow.keras.models.Model(decoder_input, decoder_output, name="decoder_mo
decoder.summary()

# Autoencoder
ae_input = tensorflow.keras.layers.Input(shape=(784), name="AE_input")
ae_encoder_output = encoder(ae_input)
ae_decoder_output = decoder(ae_encoder_output)

ae = tensorflow.keras.models.Model(ae_input, ae_decoder_output, name="AE")
ae.summary()
```

```
Model: "encoder_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
encoder_input (InputLayer)   [(None, 784)]             0         
_________________________________________________________________
encoder_dense_1 (Dense)      (None, 300)               235500    
_________________________________________________________________
encoder_leakyrelu_1 (LeakyRe (None, 300)               0         
_________________________________________________________________
encoder_dense_2 (Dense)      (None, 2)                 602       
_________________________________________________________________
encoder_output (LeakyReLU)   (None, 2)                 0         
=================================================================
Total params: 236,102
Trainable params: 236,102
```

```
Non-trainable params: 0
_________________________________________________________________
Model: "decoder_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
decoder_input (InputLayer)   [(None, 2)]               0         
_________________________________________________________________
decoder_dense_1 (Dense)      (None, 300)               900       
_________________________________________________________________
decoder_leakyrelu_1 (LeakyRe (None, 300)               0         
_________________________________________________________________
decoder_dense_2 (Dense)      (None, 784)               235984    
_________________________________________________________________
decoder_output (LeakyReLU)   (None, 784)               0         
=================================================================
Total params: 236,884
Trainable params: 236,884
Non-trainable params: 0
_________________________________________________________________
Model: "AE"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
AE_input (InputLayer)        [(None, 784)]             0         
_________________________________________________________________
encoder_model (Functional)   (None, 2)                 236102    
_________________________________________________________________
decoder_model (Functional)   (None, 784)               236884    
=================================================================
Total params: 472,986
Trainable params: 472,986
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```python
# RMSE
def rmse(y_true, y_predict):
    return tensorflow.keras.backend.mean(tensorflow.keras.backend.square(y_true-y_predi

# AE Compilation
ae.compile(loss="mse", optimizer=tensorflow.keras.optimizers.Adam(lr=0.0005))
```

In [ ]:

```python
# Preparing MNIST Dataset
(x_train_orig, y_train), (x_test_orig, y_test) = tensorflow.keras.datasets.mnist.load_d
x_train_orig = x_train_orig.astype("float32") / 255.0
x_test_orig = x_test_orig.astype("float32") / 255.0

x_train = np.reshape(x_train_orig, newshape=(x_train_orig.shape[0], np.prod(x_train_ori
x_test = np.reshape(x_test_orig, newshape=(x_test_orig.shape[0], np.prod(x_test_orig.sh
```

In [ ]:

```python
# Training AE
ae.fit(x_train, x_train, epochs=20, batch_size=256, shuffle=True, validation_data=(x_te

encoded_images = encoder.predict(x_train)
decoded_images = decoder.predict(encoded_images)
decoded_images_orig = np.reshape(decoded_images, newshape=(decoded_images.shape[0], 28,
```

```
Epoch 1/20
235/235 [==============================] - 2s 6ms/step - loss: 0.0683 - val_loss: 0.0547
Epoch 2/20
```

```
235/235 [==============================] - 1s 5ms/step - loss: 0.0544 - val_loss: 0.0530
Epoch 3/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0529 - val_loss: 0.0520
Epoch 4/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0519 - val_loss: 0.0512
Epoch 5/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0511 - val_loss: 0.0504
Epoch 6/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0505 - val_loss: 0.0499
Epoch 7/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0499 - val_loss: 0.0496
Epoch 8/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0496 - val_loss: 0.0490
Epoch 9/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0488 - val_loss: 0.0485
Epoch 10/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0485 - val_loss: 0.0481
Epoch 11/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0479 - val_loss: 0.0477
Epoch 12/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0475 - val_loss: 0.0473
Epoch 13/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0473 - val_loss: 0.0469
Epoch 14/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0468 - val_loss: 0.0466
Epoch 15/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0465 - val_loss: 0.0463
Epoch 16/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0461 - val_loss: 0.0459
Epoch 17/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0460 - val_loss: 0.0458
Epoch 18/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0458 - val_loss: 0.0455
Epoch 19/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0455 - val_loss: 0.0454
Epoch 20/20
235/235 [==============================] - 1s 5ms/step - loss: 0.0453 - val_loss: 0.0451
```

In [ ]:

```python
np.shape(x_train)
```

Out[ ]:

```
(60000, 784)
```

In [ ]:

```python
encoded_images_test = encoder.predict(x_test)
decoded_images_test = decoder.predict(encoded_images_test)
decoded_images_orig_test = np.reshape(decoded_images_test, newshape=(decoded_images_tes
```

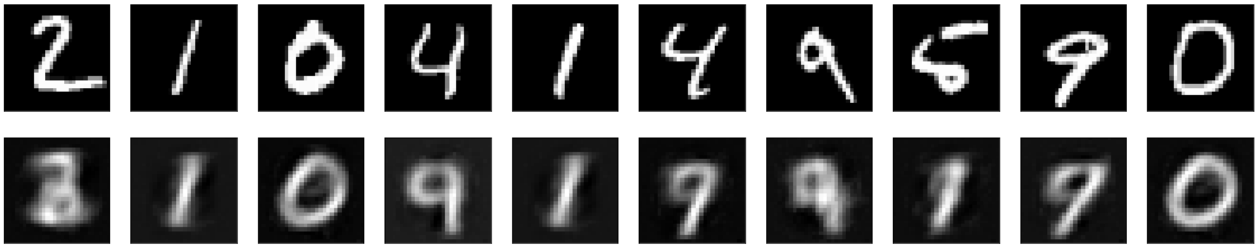
Original vs recovered images

In [ ]:

```python
num_images_to_show = 10
plt.figure(figsize=(20,4))

for i in range(1,num_images_to_show+1):
    ax = plt.subplot(2, n, i)
    plt.imshow(x_test_orig[i, :, :])
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    ax = plt.subplot(2, n, i + n)
    plt.imshow(decoded_images_orig_test[i, :, :])
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



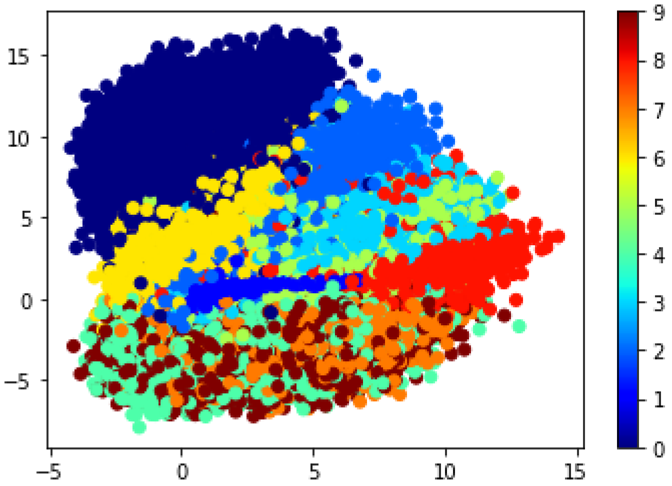
Distribution of clusters of encoded digits

In [ ]:

```
plt.figure()
plt.scatter(encoded_images[:, 0], encoded_images[:, 1], c=y_train,cmap=plt.cm.jet)
plt.colorbar()
```

Out[ ]:

```
<matplotlib.colorbar.Colorbar at 0x7f5f1d080210>
```



Convolutional Autoencoder

In [ ]:

```
import keras
from keras import layers

input_img = keras.Input(shape=(28, 28, 1))

x = layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.MaxPooling2D((2, 2), padding='same')(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = layers.MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
```

```
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = layers.UpSampling2D((2, 2))(x)
x = layers.Conv2D(16, (3, 3), activation='relu')(x)
x = layers.UpSampling2D((2, 2))(x)
decoded = layers.Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

In [ ]:

```
autoencoder.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0         
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 16)        160       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 16)        0         
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 14, 14, 8)         1160      
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 8)           0         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 7, 7, 8)           584       
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 4, 4, 8)           0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 4, 4, 8)           584       
_________________________________________________________________
up_sampling2d_3 (UpSampling2 (None, 8, 8, 8)           0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 8, 8, 8)           584       
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 16, 16, 8)         0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 14, 14, 16)        1168      
_________________________________________________________________
up_sampling2d_5 (UpSampling2 (None, 28, 28, 16)        0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 28, 28, 1)         145       
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
from keras.datasets import mnist
import numpy as np

(x_train, _), (x_test, _) = mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = np.reshape(x_train, (len(x_train), 28, 28, 1))
x_test = np.reshape(x_test, (len(x_test), 28, 28, 1))
```

In [ ]:

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=128,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Epoch 1/50
469/469 [==============================] - 6s 11ms/step - loss: 0.3099 - val_loss: 0.150
2
Epoch 2/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1436 - val_loss: 0.129
1
Epoch 3/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1272 - val_loss: 0.120
8
Epoch 4/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1196 - val_loss: 0.114
7
Epoch 5/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1151 - val_loss: 0.111
4
Epoch 6/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1121 - val_loss: 0.108
9
Epoch 7/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1097 - val_loss: 0.106
7
Epoch 8/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1076 - val_loss: 0.105
0
Epoch 9/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1062 - val_loss: 0.103
7
Epoch 10/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1048 - val_loss: 0.102
6
Epoch 11/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1037 - val_loss: 0.101
7
Epoch 12/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1027 - val_loss: 0.100
7
Epoch 13/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1015 - val_loss: 0.100
0
Epoch 14/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1009 - val_loss: 0.099
1
Epoch 15/50
469/469 [==============================] - 5s 10ms/step - loss: 0.1002 - val_loss: 0.098
6
Epoch 16/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0995 - val_loss: 0.098
1
Epoch 17/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0990 - val_loss: 0.097
8
Epoch 18/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0985 - val_loss: 0.097
4
Epoch 19/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0982 - val_loss: 0.096
8
Epoch 20/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0979 - val_loss: 0.096
```

```
2
Epoch 21/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0972 - val_loss: 0.095
9
Epoch 22/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0968 - val_loss: 0.095
8
Epoch 23/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0965 - val_loss: 0.095
3
Epoch 24/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0960 - val_loss: 0.094
6
Epoch 25/50
469/469 [==============================] - 5s 11ms/step - loss: 0.0958 - val_loss: 0.094
3
Epoch 26/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0954 - val_loss: 0.094
5
Epoch 27/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0951 - val_loss: 0.093
8
Epoch 28/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0949 - val_loss: 0.093
5
Epoch 29/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0943 - val_loss: 0.093
5
Epoch 30/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0947 - val_loss: 0.093
3
Epoch 31/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0942 - val_loss: 0.093
0
Epoch 32/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0939 - val_loss: 0.092
7
Epoch 33/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0938 - val_loss: 0.092
9
Epoch 34/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0937 - val_loss: 0.092
4
Epoch 35/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0933 - val_loss: 0.092
3
Epoch 36/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0933 - val_loss: 0.091
9
Epoch 37/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0930 - val_loss: 0.092
2
Epoch 38/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0929 - val_loss: 0.092
4
Epoch 39/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0929 - val_loss: 0.091
7
Epoch 40/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0926 - val_loss: 0.091
4
Epoch 41/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0925 - val_loss: 0.091
2
Epoch 42/50
```

```
469/469 [==============================] - 5s 10ms/step - loss: 0.0924 - val_loss: 0.091
2
Epoch 43/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0921 - val_loss: 0.091
5
Epoch 44/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0919 - val_loss: 0.090
9
Epoch 45/50
469/469 [==============================] - 5s 11ms/step - loss: 0.0919 - val_loss: 0.091
6
Epoch 46/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0918 - val_loss: 0.090
8
Epoch 47/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0915 - val_loss: 0.090
7
Epoch 48/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0914 - val_loss: 0.090
5
Epoch 49/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0916 - val_loss: 0.090
6
Epoch 50/50
469/469 [==============================] - 5s 10ms/step - loss: 0.0913 - val_loss: 0.090
3
```

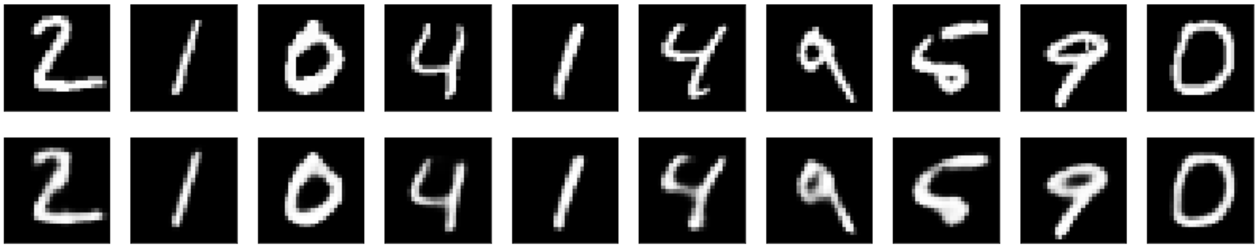
```
Out[ ]: <tensorflow.python.keras.callbacks.History at 0x7f5f70d57f50>
```

Original vs Recovered images

```
In [ ]:
decoded_imgs = autoencoder.predict(x_test)

n = 10
plt.figure(figsize=(20, 4))
for i in range(1, n + 1):
    # Display original
    ax = plt.subplot(2, n, i)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

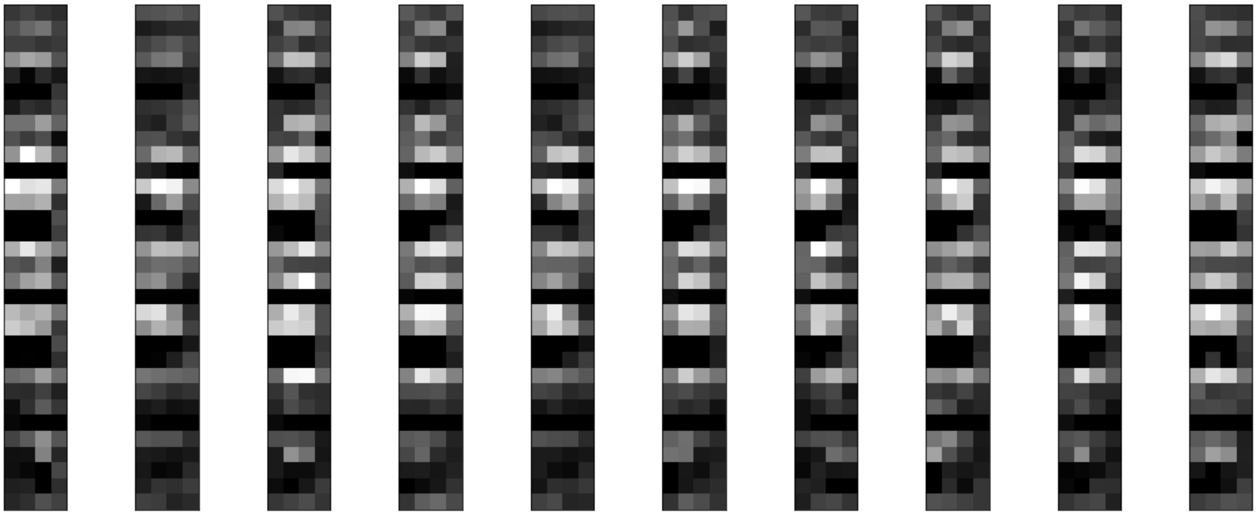


Examples of encodings of a few images

In [ ]:

```
encoder = keras.Model(input_img, encoded)
encoded_imgs = encoder.predict(x_test)

n = 10
plt.figure(figsize=(20, 8))
for i in range(1, n + 1):
    ax = plt.subplot(1, n, i)
    plt.imshow(encoded_imgs[i].reshape((4, 4 * 8)).T)
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



In [ ]:

```
encoded_imgs.shape
```

Out[ ]:

```
(10000, 4, 4, 8)
```

In [ ]: