In [1]:
```
from keras.callbacks import TensorBoard
from keras.layers import Input, Dense
from keras.models import Model
from keras.datasets import mnist
import numpy as np

(xtrain, ytrain), (xtest, ytest) = mnist.load_data()

xtrain = xtrain.astype('float32') / 255.
xtest = xtest.astype('float32') / 255.
xtrain = xtrain.reshape((len(xtrain), np.prod(xtrain.shape[1:])))
xtest = xtest.reshape((len(xtest), np.prod(xtest.shape[1:])))
xtrain.shape, xtest.shape
```

```
WARNING:tensorflow:From C:WUsersWkcosmWanaconda3WLibWsite-packagesWkerasWsrcWlosses.py:2976: The name tf.los
ses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy
instead.
```

Out[1]:
```
((60000, 784), (10000, 784))
```

Assignment

1. change the `encoding_dim` through various values (`range(2,18,2)`) and save the loss you can get. Plot the 8 pairs of dimensions vs loss on a scatter plot

In [2]:
```
#Define 'dimensions' using np.arange
dimensions = np.arange(2, 18, 2)
dimensions
```

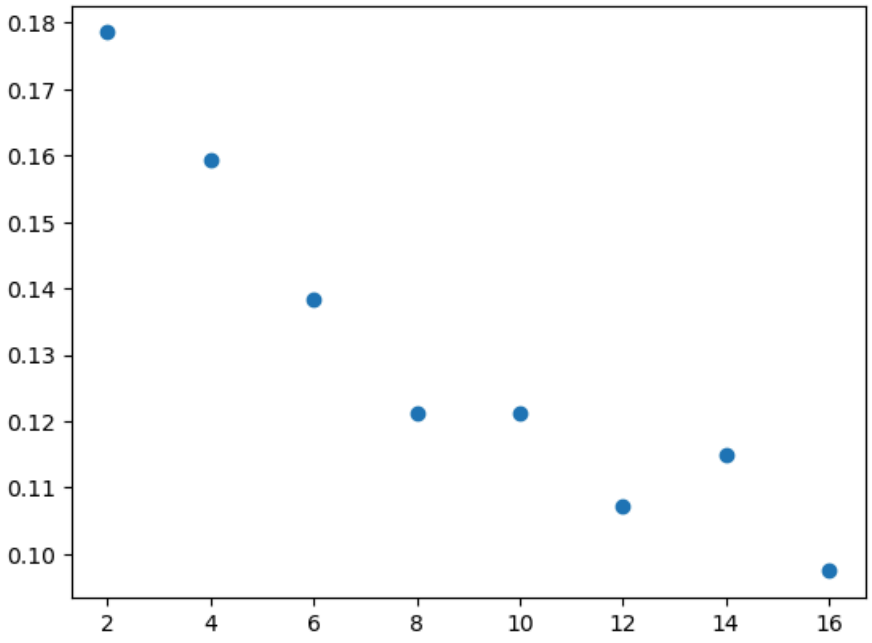
Out[2]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16])
```

In [3]:
```
losses = []

for encoding_dim in dimensions:

    # generate encoding layers
    x = input_img = Input(shape=(784,))
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    encoded = Dense(encoding_dim, activation='relu')(x)

    # generate decoding layers
    x = Dense(128, activation='relu')(encoded)
    x = Dense(256, activation='relu')(x)
    decoded = Dense(784, activation='sigmoid')(x)

    # define autoencoder (encoder and decoder is not needed in this cell)
    autoencoder = Model(input_img, decoded)

    # Compile autoendoder
    autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

    # Fit autoencoder
    autoencoder.fit(xtrain, xtrain,
                epochs=20,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest, xtest))

    # Calculate loss
    loss = autoencoder.evaluate(xtest, xtest)
    losses.append(loss)
```

```
235/235 [==============================] - 4s 17ms/step - loss: 0.1029 - val_loss: 0.1018
Epoch 12/20
235/235 [==============================] - 4s 17ms/step - loss: 0.1021 - val_loss: 0.1013
Epoch 13/20
235/235 [==============================] - 5s 20ms/step - loss: 0.1013 - val_loss: 0.1007
Epoch 14/20
235/235 [==============================] - 4s 19ms/step - loss: 0.1005 - val_loss: 0.1000
Epoch 15/20
235/235 [==============================] - 4s 19ms/step - loss: 0.1000 - val_loss: 0.0996
Epoch 16/20
235/235 [==============================] - 4s 19ms/step - loss: 0.0994 - val_loss: 0.0986
Epoch 17/20
235/235 [==============================] - 5s 20ms/step - loss: 0.0988 - val_loss: 0.0985
Epoch 18/20
235/235 [==============================] - 4s 17ms/step - loss: 0.0983 - val_loss: 0.0983
Epoch 19/20
235/235 [==============================] - 4s 16ms/step - loss: 0.0978 - val_loss: 0.0978
Epoch 20/20
235/235 [==============================] - 4s 17ms/step - loss: 0.0974 - val_loss: 0.0974
```

In [4]:
```
#Plot dimensions vs losses
import matplotlib.pyplot as plt
%matplotlib inline

plt.scatter(dimensions, losses)
```

Out[4]:
```
<matplotlib.collections.PathCollection at 0x137b49a8810>
```



Result : losses decline as dimension increases

2. *After* training an autoencoder with `encoding_dim=8`, apply noise (like the previous assignment) to *only* the input of the trained autoencoder (not the output). The output images should be without noise.

Print a few noisy images along with the output images to show they don't have noise.

In [5]:
```
#Train autoencoder with encoding_dim= 8

encoding_dim = 8

# generate encoding layers
x = input_img = Input(shape=(784,))
x = Dense(256, activation='relu')(x)
x = Dense(128, activation='relu')(x)
encoded = Dense(encoding_dim, activation='relu')(x)

# generate decoding layers
x = Dense(128, activation='relu')(encoded)
x = Dense(256, activation='relu')(x)
decoded = Dense(784, activation='sigmoid')(x)

# define autoencoder
autoencoder = Model(input_img, decoded)

# define encoder
encoder = Model(input_img, encoded)

# define decoder
encoded_input = Input(shape=(encoding_dim,))
dcd1 = autoencoder.layers[-1]
dcd2 = autoencoder.layers[-2]
dcd3 = autoencoder.layers[-3]
decoder = Model(encoded_input, dcd1(dcd2(dcd3(encoded_input))))
```

In [6]:
```
#Compile autoendoder
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

#Fit autoencoder
autoencoder.fit(xtrain, xtrain,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(xtest, xtest))
```

```
Epoch 1/50
235/235 [==============================] - 15s 20ms/step - loss: 0.2534 - val_loss: 0.1921
Epoch 2/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1823 - val_loss: 0.1730
Epoch 3/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1688 - val_loss: 0.1613
Epoch 4/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1542 - val_loss: 0.1443
Epoch 5/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1409 - val_loss: 0.1346
Epoch 6/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1328 - val_loss: 0.1296
Epoch 7/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1289 - val_loss: 0.1272
Epoch 8/50
235/235 [==============================] - 5s 22ms/step - loss: 0.1264 - val_loss: 0.1250
Epoch 9/50
235/235 [==============================] - 5s 20ms/step - loss: 0.1243 - val_loss: 0.1229
Epoch 10/50
235/235 [==============================] - 5s 21ms/step - loss: 0.1227 - val_loss: 0.1218
Epoch 11/50
235/235 [==============================] - 5s 21ms/step - loss: 0.1214 - val_loss: 0.1205
Epoch 12/50
235/235 [==============================] - 5s 23ms/step - loss: 0.1202 - val_loss: 0.1195
Epoch 13/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1192 - val_loss: 0.1185
Epoch 14/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1182 - val_loss: 0.1175
Epoch 15/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1175 - val_loss: 0.1173
Epoch 16/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1168 - val_loss: 0.1166
Epoch 17/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1161 - val_loss: 0.1159
Epoch 18/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1156 - val_loss: 0.1156
Epoch 19/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1151 - val_loss: 0.1153
Epoch 20/50
235/235 [==============================] - 5s 20ms/step - loss: 0.1146 - val_loss: 0.1149
Epoch 21/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1141 - val_loss: 0.1146
Epoch 22/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1137 - val_loss: 0.1140
Epoch 23/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1133 - val_loss: 0.1141
Epoch 24/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1130 - val_loss: 0.1136
Epoch 25/50
235/235 [==============================] - 4s 19ms/step - loss: 0.1127 - val_loss: 0.1135
Epoch 26/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1123 - val_loss: 0.1132
Epoch 27/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1120 - val_loss: 0.1131
Epoch 28/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1117 - val_loss: 0.1127
Epoch 29/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1115 - val_loss: 0.1123
Epoch 30/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1112 - val_loss: 0.1121
Epoch 31/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1109 - val_loss: 0.1122
Epoch 32/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1106 - val_loss: 0.1118
Epoch 33/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1105 - val_loss: 0.1117
Epoch 34/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1102 - val_loss: 0.1117
Epoch 35/50
235/235 [==============================] - 5s 22ms/step - loss: 0.1100 - val_loss: 0.1116
Epoch 36/50
```

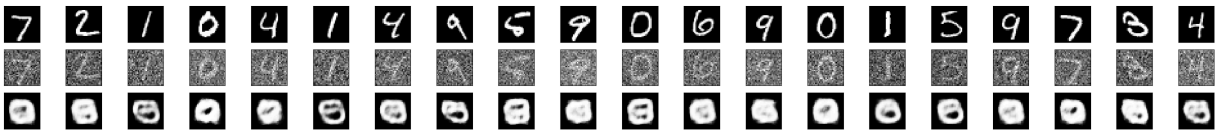
```
235/235 [==============================] - 5s 20ms/step - loss: 0.1099 - val_loss: 0.1114
Epoch 37/50
235/235 [==============================] - 5s 20ms/step - loss: 0.1096 - val_loss: 0.1116
Epoch 38/50
235/235 [==============================] - 5s 20ms/step - loss: 0.1095 - val_loss: 0.1111
Epoch 39/50
235/235 [==============================] - 6s 24ms/step - loss: 0.1093 - val_loss: 0.1110
Epoch 40/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1091 - val_loss: 0.1110
Epoch 41/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1089 - val_loss: 0.1107
Epoch 42/50
235/235 [==============================] - 4s 19ms/step - loss: 0.1088 - val_loss: 0.1106
Epoch 43/50
235/235 [==============================] - 4s 18ms/step - loss: 0.1086 - val_loss: 0.1105
Epoch 44/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1085 - val_loss: 0.1104
Epoch 45/50
235/235 [==============================] - 5s 21ms/step - loss: 0.1083 - val_loss: 0.1104
Epoch 46/50
235/235 [==============================] - 5s 22ms/step - loss: 0.1082 - val_loss: 0.1104
Epoch 47/50
235/235 [==============================] - 4s 19ms/step - loss: 0.1081 - val_loss: 0.1103
Epoch 48/50
235/235 [==============================] - 5s 23ms/step - loss: 0.1079 - val_loss: 0.1103
Epoch 49/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1078 - val_loss: 0.1100
Epoch 50/50
235/235 [==============================] - 4s 17ms/step - loss: 0.1077 - val_loss: 0.1099
```

Out[6]: `<keras.src.callbacks.History at 0x13786033550>`

In [7]:
```
#Add noise to test set (xtest2)
noise_test = np.random.normal(0.5, 0.5, (10000, 784))
xtest2 = xtest + noise_test
```

In [8]:
```
#Set input values as noised value
encoded_imgs = encoder.predict(xtest2)
decoded_imgs = decoder.predict(encoded_imgs)

# display outputs
n = 20
plt.figure(figsize=(40, 4))
for i in range(n):
    # display original test value
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(xtest[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display noised test values(input value)
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(xtest2[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display output value
    ax = plt.subplot(3, n, i + 1 + n + 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()
```

```
313/313 [==============================] - 4s 2ms/step
313/313 [==============================] - 1s 2ms/step
```



Result: output images(third row) do not have noises, but not able to distinguishable