In [ ]:
```
##### ASSIGNMENT 4 #####
```

In [ ]:
```
# Question 1 - Fully Connected Autoencoder
```

In [40]:
```
from keras import Sequential
from keras.layers import Input, Dense
from keras.models import Model
import keras
import numpy as np
import matplotlib.pyplot as plt

batch_size = 128

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scales the training and test data to range between 0 and 1.
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

input_length = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape((len(x_train), input_length))
x_test = x_test.reshape((len(x_test), input_length))
```

In [43]:
```
##Create model
model = Sequential()


# Encoder Layers
def encoder(model):
    model.add(Dense(128, input_shape=(784,), activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(2, activation='relu'))
    return model


# Decoder Layers
def decoder(encoder):
    model.add(Dense(8, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(784, activation='sigmoid'))
    return model
```

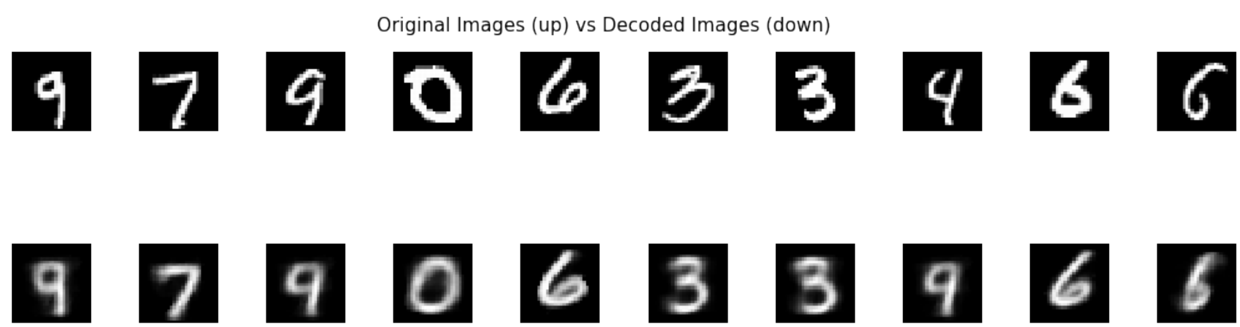
```
encoder_model = encoder(model)
decoder_model = decoder(encoder_model)
model = decoder_model

decoder_model.summary()
encoder_model.summary()
model.summary()

model.compile(optimizer='rmsprop', loss='mean_squared_error')

history = model.fit(x_train, x_train, epochs=30, batch_size=128, shuffle

#Retrieve the decoder layer from the trained model
decoder_layer1 = model.layers[-5]
decoder_layer2 = model.layers[-4]
decoder_layer3 = model.layers[-3]
decoder_layer4 = model.layers[-2]
decoder_layer5 = model.layers[-1]

# Save the decoder model
encoded_input = Input(shape=(2,))
decoder_layers = decoder_layer5(decoder_layer4(decoder_layer3(decoder_la
decoder = Model(input=encoded_input, output=decoder_layers)
decoder.summary()
decoder.save('Q1_dec_model.h5')
```

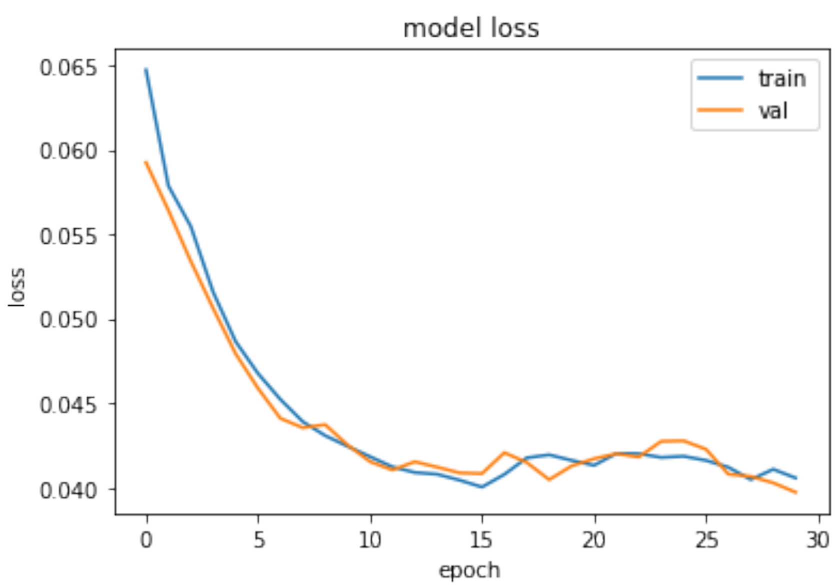
```
Epoch 15/30
60000/60000 [==============================] - 2s 25us/step - loss: 0.
0404 - val_loss: 0.0409
Epoch 16/30
60000/60000 [==============================] - 2s 25us/step - loss: 0.
0400 - val_loss: 0.0408
Epoch 17/30
60000/60000 [==============================] - 2s 28us/step - loss: 0.
0408 - val_loss: 0.0421
Epoch 18/30
60000/60000 [==============================] - 2s 28us/step - loss: 0.
0418 - val_loss: 0.0415
Epoch 19/30
60000/60000 [==============================] - 2s 27us/step - loss: 0.
0419 - val_loss: 0.0405
Epoch 20/30
60000/60000 [==============================] - 2s 27us/step - loss: 0.
0416 - val_loss: 0.0413
Epoch 21/30
60000/60000 [==============================] - 2s 26us/step - loss: 0.
```

In [44]:

```
view_decode_img = model.predict(x_test)
```

```
fig = plt.figure(figsize=(18, 4))
fig.suptitle('Original Images (up) vs Decoded Images (down)', fontsize=1
num_random_imgs = 10
random_test_images = np.random.randint(x_test.shape[0], size=num_random_

for i, image_idx in enumerate(random_test_images):
    # plot original image
    ax1, ax2 = plt.subplot(3, num_random_imgs, i + 1), plt.subplot(3, nu
    ax1.imshow(x_test[image_idx].reshape(28, 28), cmap='gray')
    ax2.imshow(view_decode_img[image_idx].reshape(28, 28), cmap='gray')
    ax1.axis('off')
    ax2.axis('off')
    plt.savefig("Q1_images.png", bbox_inches='tight')

plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()
```





In [ ]:
```
# Question 2 - Convolutional Autoencoder
```

In [ ]:
```
# Using a minimal bottleneck with 2 neurons
```

In [45]:
```
from keras import losses, regularizers, optimizers, Sequential
from keras.models import Model
from keras.layers import Convolution2D, Conv2DTranspose, Input, Dense, M
    ZeroPadding2D, Cropping2D, Reshape, Flatten
import matplotlib.pyplot as plt
import numpy as np
import keras

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scales the training and test data to range between 0 and 1.
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0


input_length = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape((len(x_train), 28, 28, 1))
x_test = x_test.reshape((len(x_test), 28, 28, 1))
```

In [46]:
```
model = Sequential()

# Encoder Layers
model.add(Conv2D(16, (3, 3), input_shape=x_train.shape[1:], activation='
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(2, (3, 3), strides=(2,2), activation='relu', padding='s
model.add(MaxPooling2D((2, 2), padding='same'))

model.summary()

# Decoder Layers
model.add(Conv2D(2, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
```

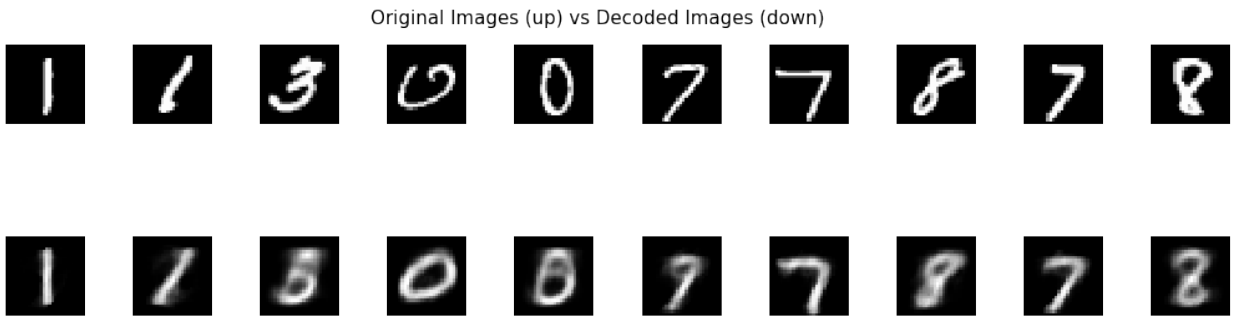
```
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(1, (3, 3), activation='sigmoid', padding='same'))

model.summary()


model.compile(optimizer='rmsprop', loss='mean_squared_error')
history = model.fit(x_train, x_train, epochs=30, batch_size=128, validat
```

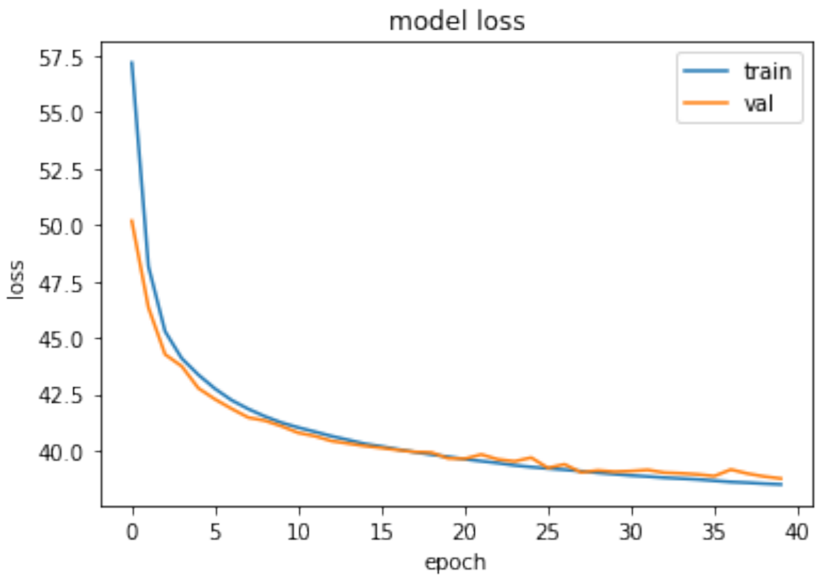
```
0.0537 - val_loss: 0.0539
Epoch 15/30
60000/60000 [==============================] - 35s 579us/step - loss:
0.0535 - val_loss: 0.0528
Epoch 16/30
60000/60000 [==============================] - 34s 574us/step - loss:
0.0533 - val_loss: 0.0542
Epoch 17/30
60000/60000 [==============================] - 35s 578us/step - loss:
0.0531 - val_loss: 0.0530
Epoch 18/30
60000/60000 [==============================] - 34s 574us/step - loss:
0.0529 - val_loss: 0.0523
Epoch 19/30
60000/60000 [==============================] - 35s 577us/step - loss:
0.0526 - val_loss: 0.0525
Epoch 20/30
60000/60000 [==============================] - 35s 577us/step - loss:
0.0525 - val_loss: 0.0523
```

In [186]:

```
view_decode_img = model.predict(x_test)

fig = plt.figure(figsize=(18, 4))
fig.suptitle('Original Images (up) vs Decoded Images (down)', fontsize=1
num_random_imgs = 10
random_test_images = np.random.randint(x_test.shape[0], size=num_random_

for i, image_idx in enumerate(random_test_images):
    # plot original image
    ax1, ax2 = plt.subplot(3, num_random_imgs, i + 1), plt.subplot(3, nu
    ax1.imshow(x_test[image_idx].reshape(28, 28), cmap='gray')
    ax2.imshow(view_decode_img[image_idx].reshape(28, 28), cmap='gray')
    ax1.axis('off')
    ax2.axis('off')
    plt.savefig("Q2.1_images.png", bbox_inches='tight')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

```python
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()
```





In [ ]:
```python
# Question 2.2 - Convolutional Autoencoder
```

In [ ]:
```python
# Using a larger bottleneck with 6 neurons
```

In [48]:
```
from keras import losses, regularizers, optimizers, Sequential
from keras.models import Model
from keras.layers import Convolution2D, Conv2DTranspose, Input, Dense, M
    ZeroPadding2D, Cropping2D, Reshape, Flatten
import matplotlib.pyplot as plt
import numpy as np
import keras

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scales the training and test data to range between 0 and 1.
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0


input_length = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape((len(x_train), 28, 28, 1))
x_test = x_test.reshape((len(x_test), 28, 28, 1))
```

In [49]:
```

model = Sequential()

# Encoder Layers
model.add(Conv2D(16, (3, 3), input_shape=x_train.shape[1:], activation='
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D((2, 2), padding='same'))
model.add(Conv2D(6, (3, 3), strides=(2,2), activation='relu', padding='s
model.add(MaxPooling2D((2, 2), padding='same'))

model.summary()

# Decoder Layers
model.add(Conv2D(6, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(16, (3, 3), activation='relu', padding='same'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(16, (3, 3), activation='relu'))
model.add(UpSampling2D((2, 2)))
model.add(Conv2D(1, (3, 3), activation='sigmoid', padding='same'))
```

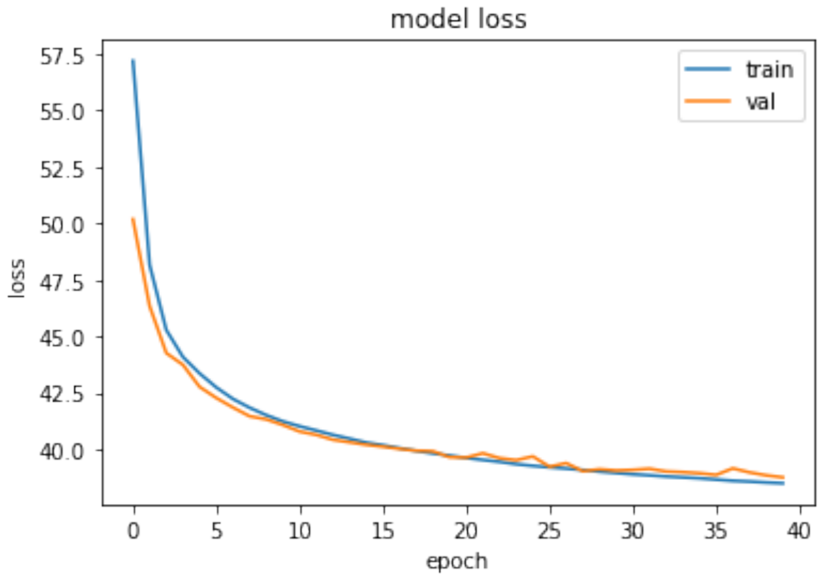
```
model.summary()

model.compile(optimizer='rmsprop', loss='mean_squared_error')
history = model.fit(x_train, x_train, epochs=30, batch_size=128, validat
```

```
0.0396 - val_loss: 0.0384
Epoch 15/30
60000/60000 [==============================] - 36s 599us/step - loss:
0.0391 - val_loss: 0.0390
Epoch 16/30
60000/60000 [==============================] - 36s 600us/step - loss:
0.0386 - val_loss: 0.0401
Epoch 17/30
60000/60000 [==============================] - 36s 601us/step - loss:
0.0383 - val_loss: 0.0377
Epoch 18/30
60000/60000 [==============================] - 36s 600us/step - loss:
0.0379 - val_loss: 0.0370
Epoch 19/30
60000/60000 [==============================] - 36s 601us/step - loss:
0.0376 - val_loss: 0.0375
Epoch 20/30
60000/60000 [==============================] - 36s 601us/step - loss:
0.0373 - val_loss: 0.0377
```

In [165]:

```
view_decode_img = model.predict(x_test)

fig = plt.figure(figsize=(18, 4))
fig.suptitle('Original Images (up) vs Decoded Images (down)', fontsize=1
num_random_imgs = 10
random_test_images = np.random.randint(x_test.shape[0], size=num_random_

for i, image_idx in enumerate(random_test_images):
    # plot original image
    ax1, ax2 = plt.subplot(3, num_random_imgs, i + 1), plt.subplot(3, nu
    ax1.imshow(x_test[image_idx].reshape(28, 28), cmap='gray')
    ax2.imshow(view_decode_img[image_idx].reshape(28, 28), cmap='gray')
    ax1.axis('off')
    ax2.axis('off')
    plt.savefig("Q2_images.png", bbox_inches='tight')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()
```

Original Images (up) vs Decoded Images (down)



In [ ]:
```
# Question 3a - Visualize trained autoencoder from Q1 through generated
```

In [170]:
```
#### from keras.models import load_model
import matplotlib.pyplot as plt
import numpy as np

decoder = load_model('Q1_dec_model.h5')
decoder.summary()


plt.figure(figsize=(18, 4))
num_images = 10
random_test_images = np.asarray([np.random.normal(0, 5.0, 10), np.random
random_test_images = random_test_images.reshape((10, 2))
view_decode_imgs = decoder.predict(random_test_images)


for i in range(len(random_test_images)):
    ax = plt.subplot(3, num_images, 2 * num_images + i + 1)
    plt.imshow(view_decode_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
Model: "model_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_13 (InputLayer)        (None, 2)                 0
_________________________________________________________________
dense_92 (Dense)             (None, 8)                 24
_________________________________________________________________
dense_93 (Dense)             (None, 32)                288
_________________________________________________________________
dense_94 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_95 (Dense)             (None, 128)               8320
_________________________________________________________________
dense_96 (Dense)             (None, 784)               101136
=================================================================
Total params: 111,880
Trainable params: 111,880
Non-trainable params: 0
_________________________________________________________________
```

In [148]:
```
# Question 3b - Autoencoder with standard multi-variate normal distribut
```

In [113]:
```
from keras import Sequential
from keras.layers import Input, Dense, BatchNormalization
from keras.models import Model
import keras
import numpy as np
import matplotlib.pyplot as plt

batch_size = 128
epochs = 50

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scales the training and test data to range between 0 and 1.
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0


input_length = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape((len(x_train), input_length))
x_test = x_test.reshape((len(x_test), input_length))
```

In [53]:
```
##Create model
model = Sequential()


# Encoder Layers
def encoder(model):
    model.add(Dense(128, input_shape=(784,), activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(2, activation='relu'))
    model.add(BatchNormalization(beta_initializer='zeros', gamma_initial
                                 moving_variance_initializer='ones'))
    return model


# Decoder Layers
def decoder(encoder):
    model.add(Dense(8, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(784, activation='sigmoid'))
    return model


encoder_model = encoder(model)
decoder_model = decoder(encoder_model)
```

```
model = decoder_model

decoder_model.summary()
encoder_model.summary()
model.summary()

model.compile(optimizer='rmsprop', loss='mean_squared_error')

history = model.fit(x_train, x_train, epochs=30, batch_size=128, shuffle

#Retrieve the decoder layer from the trained model
decoder_layer1 = model.layers[-5]
decoder_layer2 = model.layers[-4]
decoder_layer3 = model.layers[-3]
decoder_layer4 = model.layers[-2]
decoder_layer5 = model.layers[-1]

# Save the decoder model
encoded_input = Input(shape=(2,))
decoder_layers = decoder_layer5(decoder_layer4(decoder_layer3(decoder_la
decoder = Model(input=encoded_input, output=decoder_layers)
decoder.summary()
decoder.save('Q3b_dec_model.h5')

view_decode_img = model.predict(x_test)
```

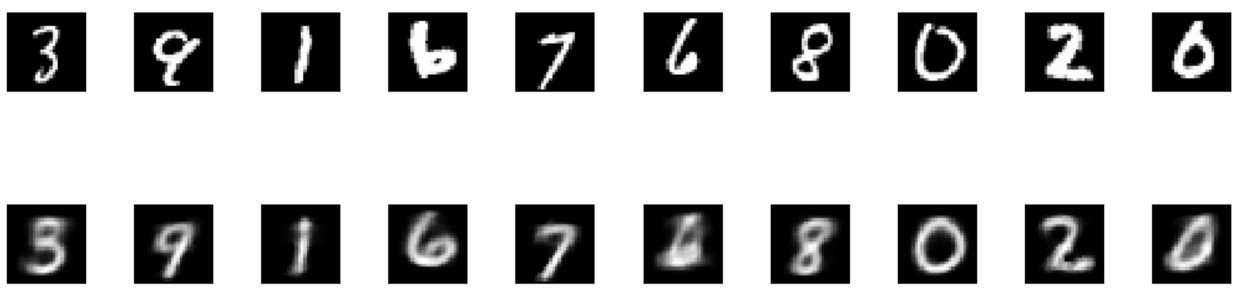
```
Epoch 15/30
60000/60000 [==============================] - 2s 28us/step - loss: 0.
0410 - val_loss: 0.0392
Epoch 16/30
60000/60000 [==============================] - 2s 29us/step - loss: 0.
0407 - val_loss: 0.0388
Epoch 17/30
60000/60000 [==============================] - 2s 28us/step - loss: 0.
0405 - val_loss: 0.0391
Epoch 18/30
60000/60000 [==============================] - 2s 28us/step - loss: 0.
0405 - val_loss: 0.0386
Epoch 19/30
60000/60000 [==============================] - 2s 28us/step - loss: 0.
0405 - val_loss: 0.0383
Epoch 20/30
60000/60000 [==============================] - 2s 28us/step - loss: 0.
0402 - val_loss: 0.0383
Epoch 21/30
60000/60000 [==============================] - 2s 29us/step - loss: 0.
```

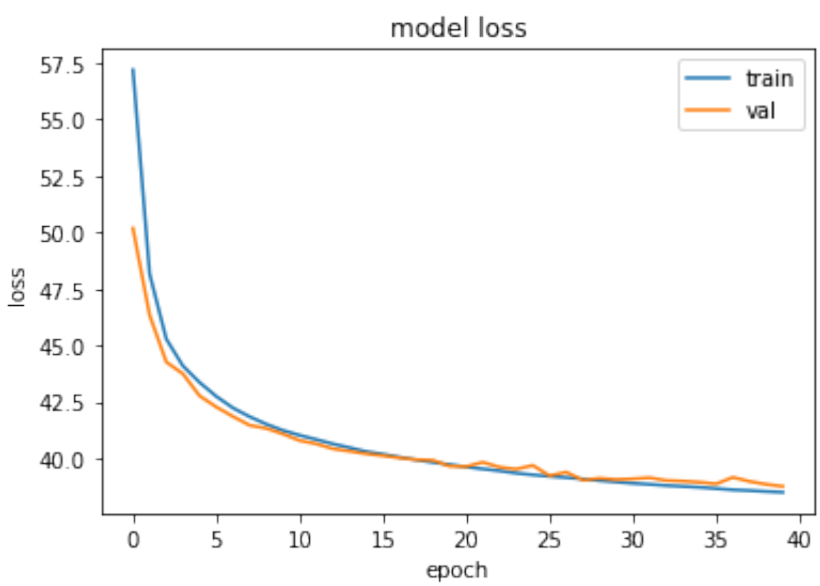
In [174]:

```
fig = plt.figure(figsize=(18, 4))
fig.suptitle('Original Images (up) vs Decoded Images (down)', fontsize=1
num_random_imgs = 10
random_test_images = np.random.randint(x_test.shape[0], size=num_random_
```

```
for i, image_idx in enumerate(random_test_images):
    # plot original image
    ax1, ax2 = plt.subplot(3, num_random_imgs, i + 1), plt.subplot(3, nu
    ax1.imshow(x_test[image_idx].reshape(28, 28), cmap='gray')
    ax2.imshow(view_decode_img[image_idx].reshape(28, 28), cmap='gray')
    ax1.axis('off')
    ax2.axis('off')
    plt.savefig("Q3b_images.png", bbox_inches='tight')

plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()
```

Original Images (up) vs Decoded Images (down)

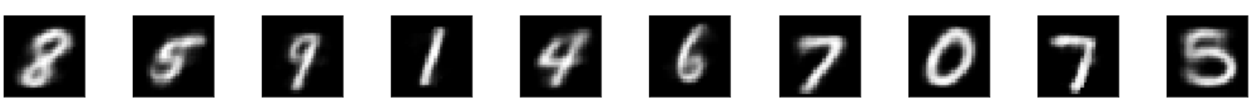





```
In [ ]: #Question 3b (2) - Randomly generate inputs to the bottleneck layer that
        #multi-variate standard normal distribution
```

In [176]:
```
from keras.models import load_model
import matplotlib.pyplot as plt
import numpy as np

decoder = load_model('Q3b_dec_model.h5')
decoder.summary()


plt.figure(figsize=(18, 4))
num_images = 10
random_test_images = np.asarray([np.random.normal(0, 1.0, 10), np.random
random_test_images = random_test_images.reshape((10, 2))
view_decode_imgs = decoder.predict(random_test_images)


for i in range(len(random_test_images)):
    ax = plt.subplot(3, num_images, 2 * num_images + i + 1)
    plt.imshow(view_decode_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
Model: "model_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        (None, 2)                 0
_________________________________________________________________
dense_102 (Dense)            (None, 8)                 24
_________________________________________________________________
dense_103 (Dense)            (None, 32)                288
_________________________________________________________________
dense_104 (Dense)            (None, 64)                2112
_________________________________________________________________
dense_105 (Dense)            (None, 128)               8320
_________________________________________________________________
dense_106 (Dense)            (None, 784)               101136
=================================================================
Total params: 111,880
Trainable params: 111,880
Non-trainable params: 0
_________________________________________________________________
```



In [ ]:
```
#3b (2)-

# As you can see from the above experimentation, the results are better
# Batch Normalization at the end of the encoder layer. The Batch Normali
# value 1 is passed for multi variate normal distribution.
```

In [177]:
```
                                            # Question 3c

# Are the output images different between 1) and 2)? If so, why do you t

                                            # Answer

# The main difference is that 3b gives much better readable images than

# 3b is better because we used the Batch Normalization with multi variat
# process, the encoder is normalized with the range of 0 to 1. Thus whil
# to converge easily and provide us with much better results. (i.e it is
# value)

# Whereas for 3a, the encoder output is not in range of 0 to 1. Thus fee
# distribution gives a slightly less quality and less range of images. T
# above the range that we provided. This gives us a restriction of the i

# But this is not case for 3b, while we try to input random numbers in t
# much better as we have trained the decoder in the same expected range
# readable images

# Also, Batch Normalization is usually used to obtain a steady distribut
# We used it at the end of the encoder layer, so that the convergence is
# sensitive to changes in the distribution of the inputs, or the hidden
# faster and a better result.
```

In [7]:
```
# Question 4 - Variational Autoencoder
```

In [64]:
```
from keras import Sequential
from keras.layers import Input, Dense, Lambda
from keras.models import Model
from keras.datasets import mnist
import tensorflow as tf
from keras.losses import mse
from keras import backend as K
import numpy as np
import matplotlib.pyplot as plt

latent_dimension = 2

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Normalizing the dataset
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

input_length = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape((len(x_train), input_length))
x_test = x_test.reshape((len(x_test), input_length))

input_shape = Input(shape=(input_length,))
```

In [65]:
```
##Create model
model = Sequential()


def sampling(args):
    mu_vector, sigma_vector = args
    epsilon = tf.random_normal(K.shape(sigma_vector), dtype=np.float32, 
    sample_latent_vec = mu_vector + K.exp(0.5*sigma_vector) * epsilon
    return sample_latent_vec




def encoder(inputs):
    x_encoded = Dense(256, activation='relu')(inputs)
    #x_encoded = Dense(128, activation='relu')(x_encoded)
    x_encoded = Dense(2, activation='relu')(x_encoded)
    #x_encoded = BatchNormalization(beta_initializer='zeros', gamma_init
     #                               moving_variance_initializer='ones')(

    mu_vector = Dense(latent_dimension)(x_encoded)
    sigma_vector = Dense(latent_dimension)(x_encoded)
    encoded = Lambda(sampling, output_shape=(latent_dimension,))([mu_vec
    return encoded, mu_vector, sigma_vector
```

```
# decoder
def decoder(encoded):
    #z_decoder1 = Dense(128, activation='relu')
    decoded = Dense(256, activation='relu')
    x_decoded = Dense(x_train.shape[1], activation='sigmoid')

    #z_decoded = z_decoder1(z)
    decoded = decoded(encoded)
    decoded = x_decoded(decoded)
    return decoded


# VAE model
encoded, mu_vector, sigma_vector = encoder(input_shape)
outputs = decoder(encoded)
model = Model(input_shape, outputs)


### Finding the VAE Loss
def calc_vae_loss(inputs, outputs):
    reconstruction_loss = mse(inputs, outputs) * x_train.shape[1]
    # KL Divergence
    kl_loss = 1 + sigma_vector - K.square(mu_vector) - K.exp(sigma_vecto
    kl_loss = -0.5 * tf.reduce_sum(kl_loss, axis=-1)
    vae_loss = tf.reduce_mean(reconstruction_loss + kl_loss)
    return vae_loss


vae_loss = calc_vae_loss(input_shape, outputs)
model.add_loss(vae_loss)
model.compile(optimizer='rmsprop')
model.summary()

history = model.fit(x_train, epochs=40, batch_size=128, validation_data=
```

```
Epoch 14/40
60000/60000 [==============================] - 2s 30us/step - loss: 40
.4772 - val_loss: 40.3188
Epoch 15/40
60000/60000 [==============================] - 2s 32us/step - loss: 40
.3025 - val_loss: 40.2088
Epoch 16/40
60000/60000 [==============================] - 2s 30us/step - loss: 40
.1880 - val_loss: 40.1228
Epoch 17/40
60000/60000 [==============================] - 2s 30us/step - loss: 40
.0576 - val_loss: 40.0275
Epoch 18/40
60000/60000 [==============================] - 2s 30us/step - loss: 39
.9393 - val_loss: 39.9447
Epoch 19/40
```

```
60000/60000 [==============================] - 2s 31us/step - loss: 39
.8255 - val_loss: 39.9206
Epoch 20/40
60000/60000 [==============================] - 2s 31us/step - loss: 39
```

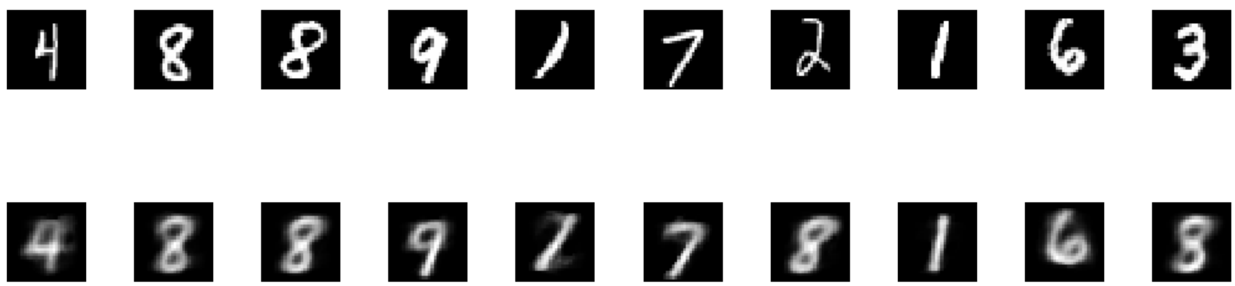
In [71]:

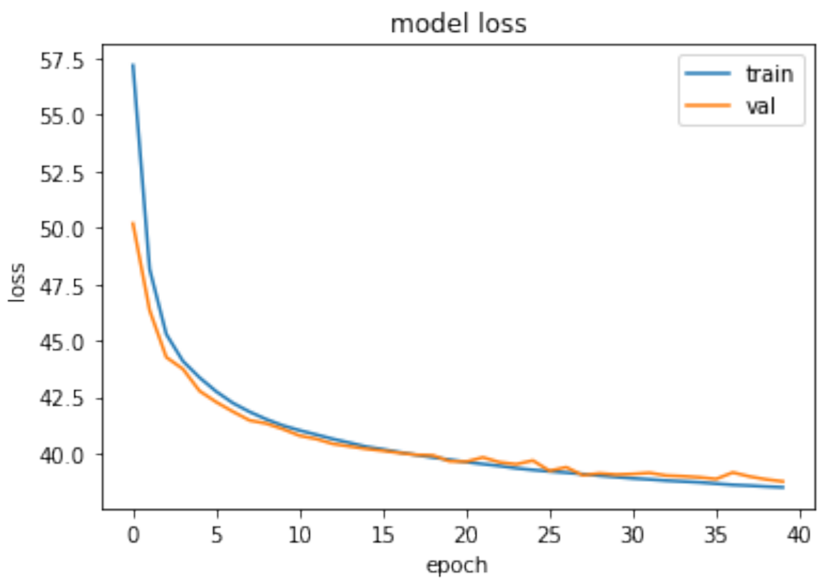
```
# Retrieve the decoder layer from the trained model
#decoder_layer1 = model.layers[-3]
decoder_layer1 = model.layers[-2]
decoder_layer2 = model.layers[-1]
# Save the decoder model
encoded_input = Input(shape=(2,))
decoder_layers = decoder_layer2(decoder_layer1(encoded_input))
decoder = Model(input=encoded_input, output=decoder_layers)
decoder.summary()
decoder.save('Q4_dec_model.h5')

predict_decode_img = model.predict(x_test)

fig = plt.figure(figsize=(18, 4))
fig.suptitle('Original Training Images (up) vs Decoded Testing Images (d
num_random_imgs = 10
random_test_images = np.random.randint(x_test.shape[0], size=num_random_

for i, image_idx in enumerate(random_test_images):
    # plot original image
    ax1, ax2 = plt.subplot(3, num_random_imgs, i + 1), plt.subplot(3, nu
    ax1.imshow(x_test[image_idx].reshape(28, 28), cmap='gray')
    ax2.imshow(predict_decode_img[image_idx].reshape(28, 28), cmap='gray
    ax1.axis('off')
    ax2.axis('off')
    plt.savefig("Q4_images.png", bbox_inches='tight')

plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'])
plt.show()
```

```
/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:8: Us
erWarning: Update your `Model` call to the Keras 2 API: `Model(inputs=
Tensor("in..., outputs=Tensor("de...)`


Model: "model_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
input_21 (InputLayer)        (None, 2)                 0
_________________________________________________________________
dense_129 (Dense)            (None, 256)               768
_________________________________________________________________
dense_130 (Dense)            (None, 784)               201488
=================================================================
Total params: 202,256
Trainable params: 202,256
Non-trainable params: 0
_________________________________________________________________
```

Original Training Images (up) vs Decoded Testing Images (down)

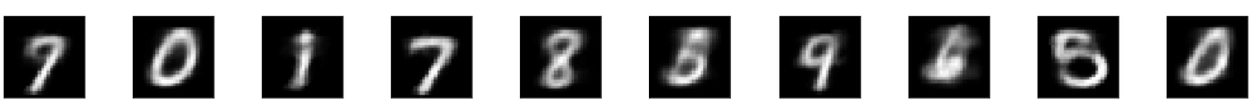




In [ ]:
```
# Q4 - Visualizing the random generated numbers from the trained vae mod
```

In [183]:
```
from keras.models import load_model
import matplotlib.pyplot as plt
import numpy as np

decoder = load_model('Q4_dec_model.h5')
decoder.summary()


plt.figure(figsize=(18, 4))
num_images = 10
random_test_images = np.asarray([np.random.normal(0, 1.0, 10), np.random
random_test_images = random_test_images.reshape((10, 2))
view_decode_imgs = decoder.predict(random_test_images)


for i in range(len(random_test_images)):
    ax = plt.subplot(3, num_images, 2 * num_images + i + 1)
    plt.imshow(view_decode_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
Model: "model_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_21 (InputLayer)        (None, 2)                 0         
_________________________________________________________________
dense_129 (Dense)            (None, 256)               768       
_________________________________________________________________
dense_130 (Dense)            (None, 784)               201488    
=================================================================
Total params: 202,256
Trainable params: 202,256
Non-trainable params: 0
_________________________________________________________________
```



In [ ]:
```
                                                # Question 4

# Does the VAE produce a different quality of output image?

                                                # Answer

# VAE consists of both an encoder and a decoder and that is trained to m
# encoded-decoded data and the initial data. Instead of encoding an inpu
# the latent space.We are optimizing both the reconstruction loss and th
# the loss and to give much better results. As expected, the results wer
# experimentations.

# I attempted few of the experiments myself during this process by tryin
# adam. Adding regularizers to avoid overfitting etc. During all the abo
# (with proper optimization & fine tuning) compared to all the above aut
```