In [0]:
```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import backend as K
import pickle
```

In [0]:
```python
# Check that TF 2.1.0 is in use
# I use colab and accelerate by their GPU so tf is 2.2.0
print(tf.__version__)
```

```
2.2.0
```

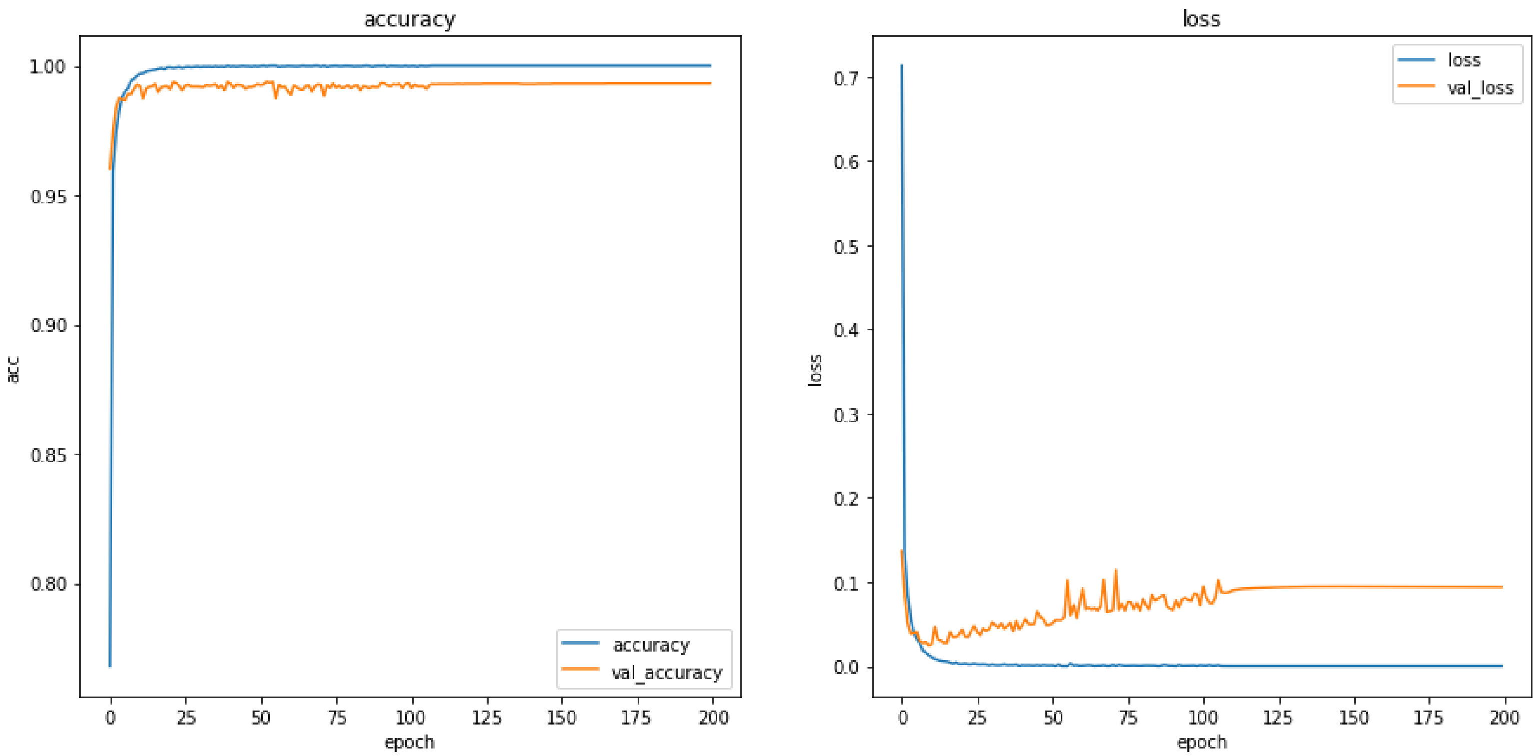
Points awarded for correct working models, questions, and plots.

[+10 per model correct and working -5 for failure on either]

1.1 Answer the following questions:

1. Explain the indication of overfitting and how this occurs (provide plot supporting your answer)? [+4 answer, +4 plot, +2 answer and plot agree]

The **indication** of overfitting is that when our model does much better on the training set than on the test/validation set, then we're likely overfitting. Just like the figure below, we can see that the accuracy of the training dataset is higher than of the validation set and the loss on the validation set goes much higher then on the training set, that means the performance of our model is only good on the training set and it can be overfitting.



The **reason** for the overfitting is that: when the algorithm is too complicated or flexible but the training data is limited, it can end up "memorizing the training dataset" instead of finding the features, so that the model will make predictions based on the relationship between inputs and labels in the training dataset and perform unusually well on its training data but badly on the validation set, then we can see a large difference of the curves in this figure.

2. Explain how overfit can hinder performance of a model when deployed. [+6 answer]

Just as explained in problem1.1.1, overfitting happens when the algorithm is too complicated but the training data is limited and leads to an unusually good performance on the training set but may fail to fit additional data, and this may affect the accuracy of predicting future observations. So when we deployed this kind of overfitting model to the real test case, the output accuracy maybe not very good and the performance may not be very robust.

3. Name two ways to avoid this. [+2 answer, +2 answer]

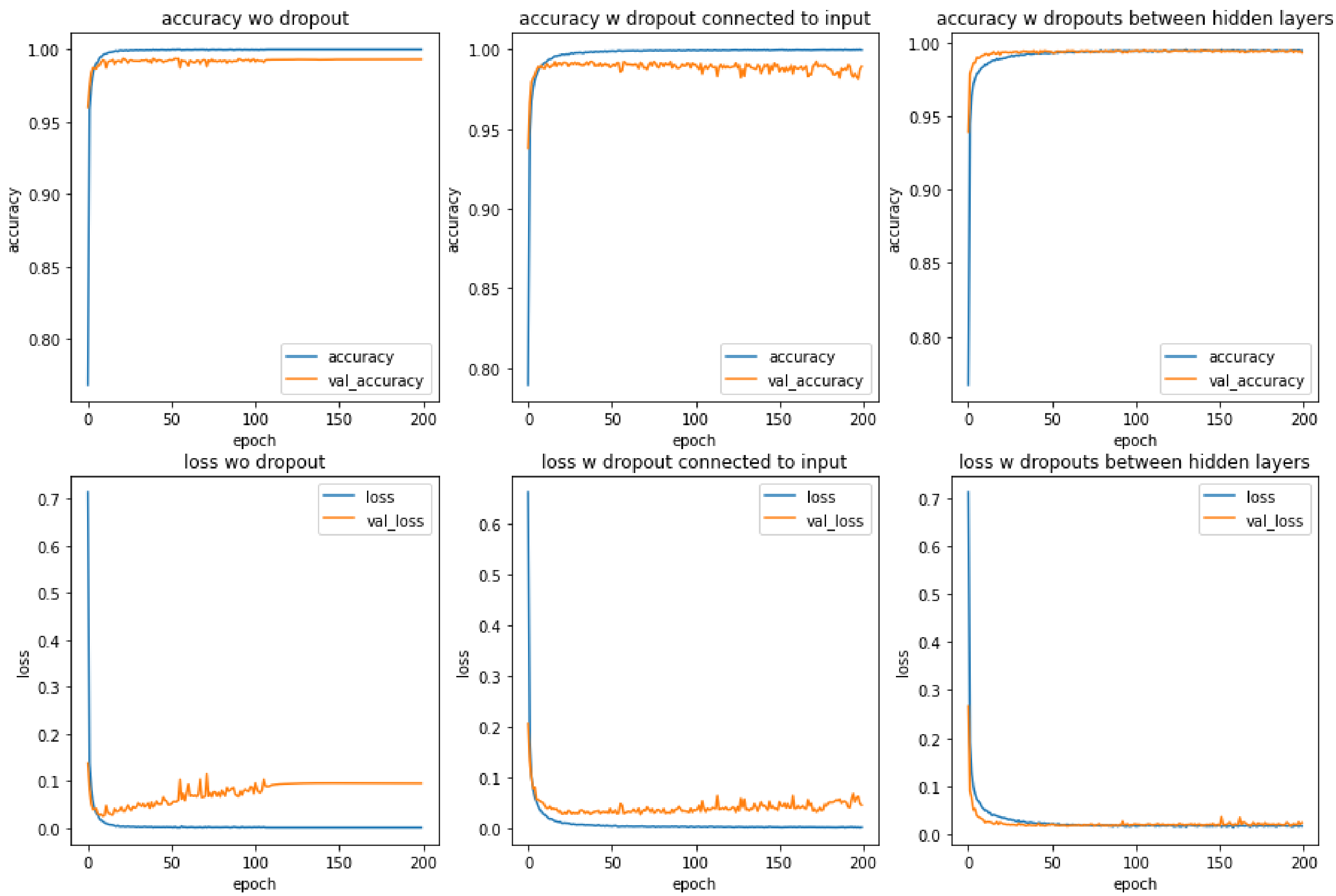
1. We can train our model with more data to prevent overfitting and add more diversity to the training set.
2. We can do the data simplification. Since the overfitting can occur due to the complexity of a model, we can try to decrease the complexity of the model to make it simple enough that it does not overfit. We can achieve that by pruning a decision tree, reducing the number of parameters in a neural network, and using dropout on a neutral network.

1.2 Answer the following question:

1. Explain how dropout affected your loss (provide plot supporting your answer). [+5 answer, +5 plot]

Here I did two experiments according to the requirement.

1. The first one is that a dropout of 30% was inserted between the input and the first hidden layer. Then the model was trained again and the final accuracy result is **98.92%**. Compared with the accuracy score without the dropout layer(**98.32%**), the accuracy decreases a little. **The second column in the figure below shows the accuracy and loss change during the training process under this case, and we can see that the overfitting situation did not improve too much and the accuracy curve even became worse.**
2. The second experiment is that the dropout between input and hidden was removed and a dropout to each hidden layer except between softmax and output layer was added. The final accuracy score is **98.37%**, better than the result in the first experiment and the result without dropout. **From the third column of the figure below, we can see that the curves of the training set and validation set are quite similar no matter from the accuracy plot or from the loss plot, which means that the overfitting problem was solved.**



The **reason** for this phenomenon is that the dropout actually is trying to ignore units (i.e., neurons) during the training phase of a certain set of neurons which is chosen at random, the ignoring units are not considered during a particular forward or backward pass, so it is a kind of data simplification and can help to overcome the overfitting. However, if the dropout layer is inserted between the input layer and the first hidden layer, the input space information can be broken and some information from this raw input can be ignored, which may reduce the performance of the model and results to a lower accuracy(just as the result shown in the second column). So we need to avoid doing that and only apply the dropout between other layers, then we can see a good result and solve the overfitting problem.

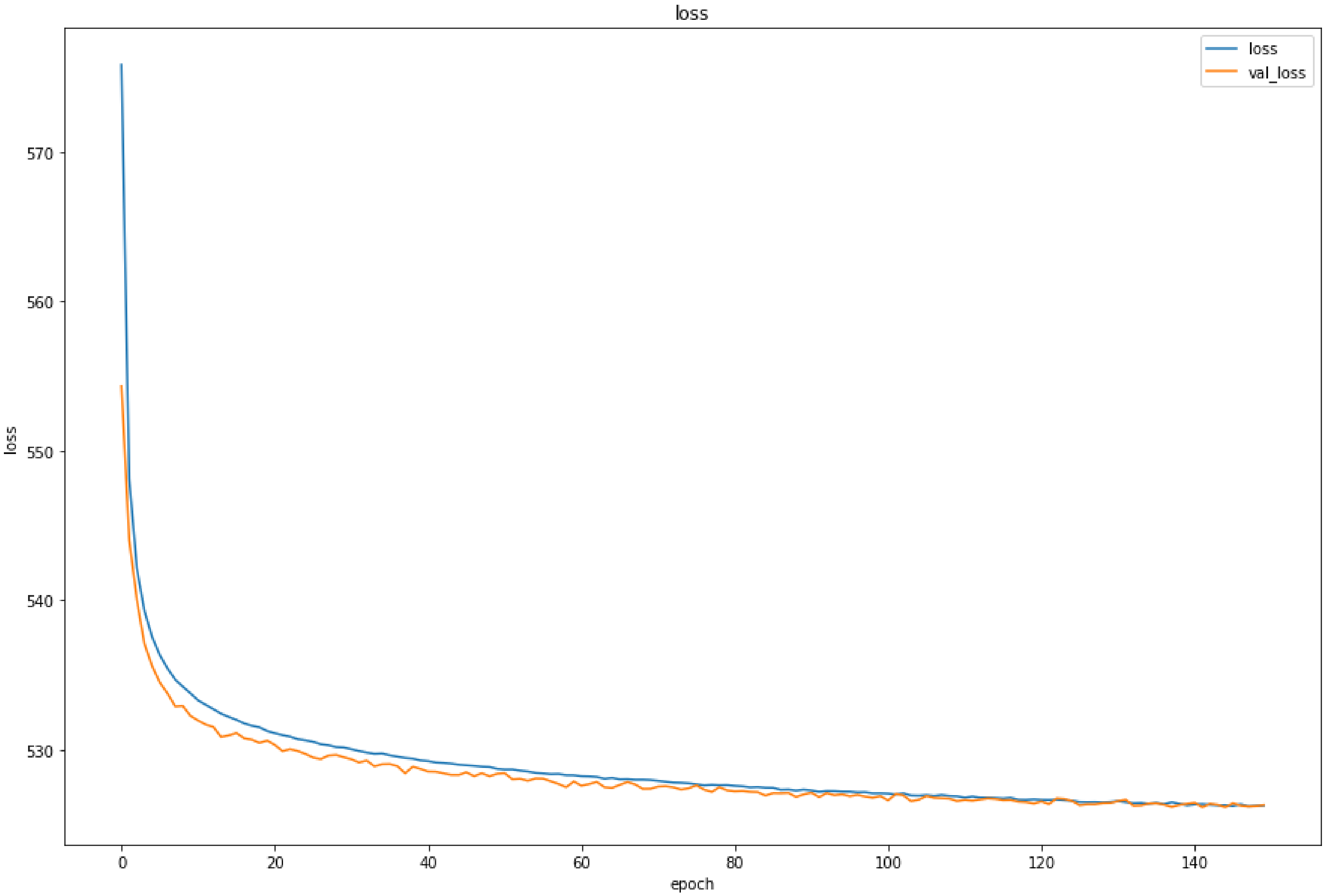
Bonus Answer the following question:

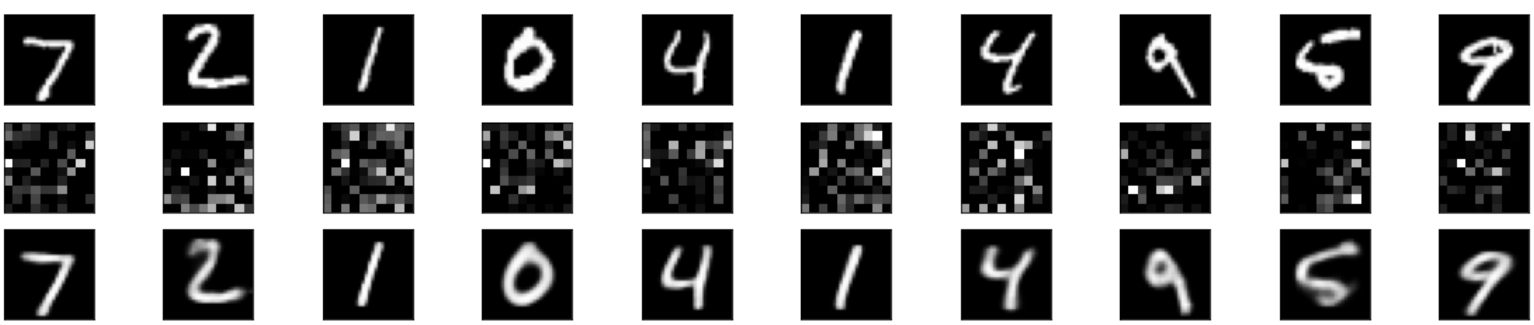
1. Considering that encoder and decoder can be constructed as separate components, trained as a single unit, and then separated for use . What uses can you brainstorm? [+5 bonus makeup points]

From my perspective, I think the separated autoencoder can be used in the encryption field like **secure communication, image encryption** ...

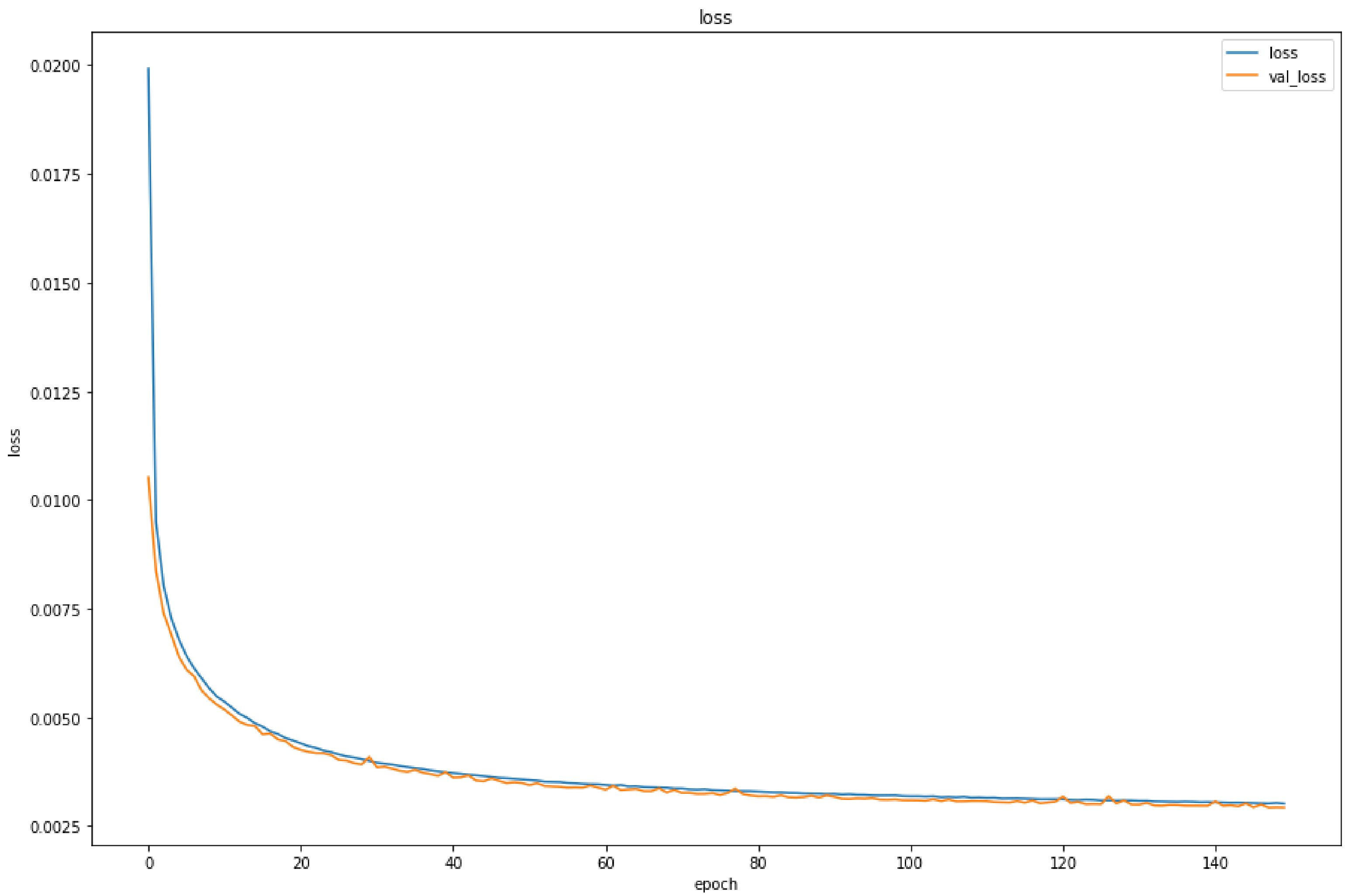
Say for example, when we need to do the secure communication, we can train an autoencoder first and then separated the model, and the sender can keep the encoder part and input the message into the encoder then send the output of the encoder. When the receiver gets the signal, the raw message can be recovered by the decoder. I think compared with the other secure communication method, use encoder and decoder as the key can be more secure, but the accuracy may be the main limitation. Probably an overfitting model can be better for this application. Also, the image encryption is similar, we can train AE first use the encoder and decoder as the keys.

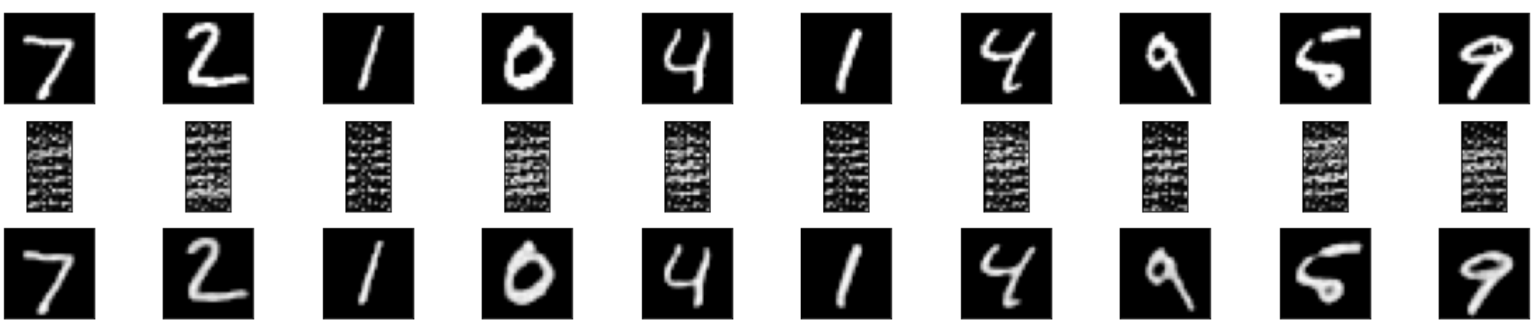
2.1 Linear AE points for constructed model, no questions here.





2.2 Convolutional AE points for constructed model, no questions here.



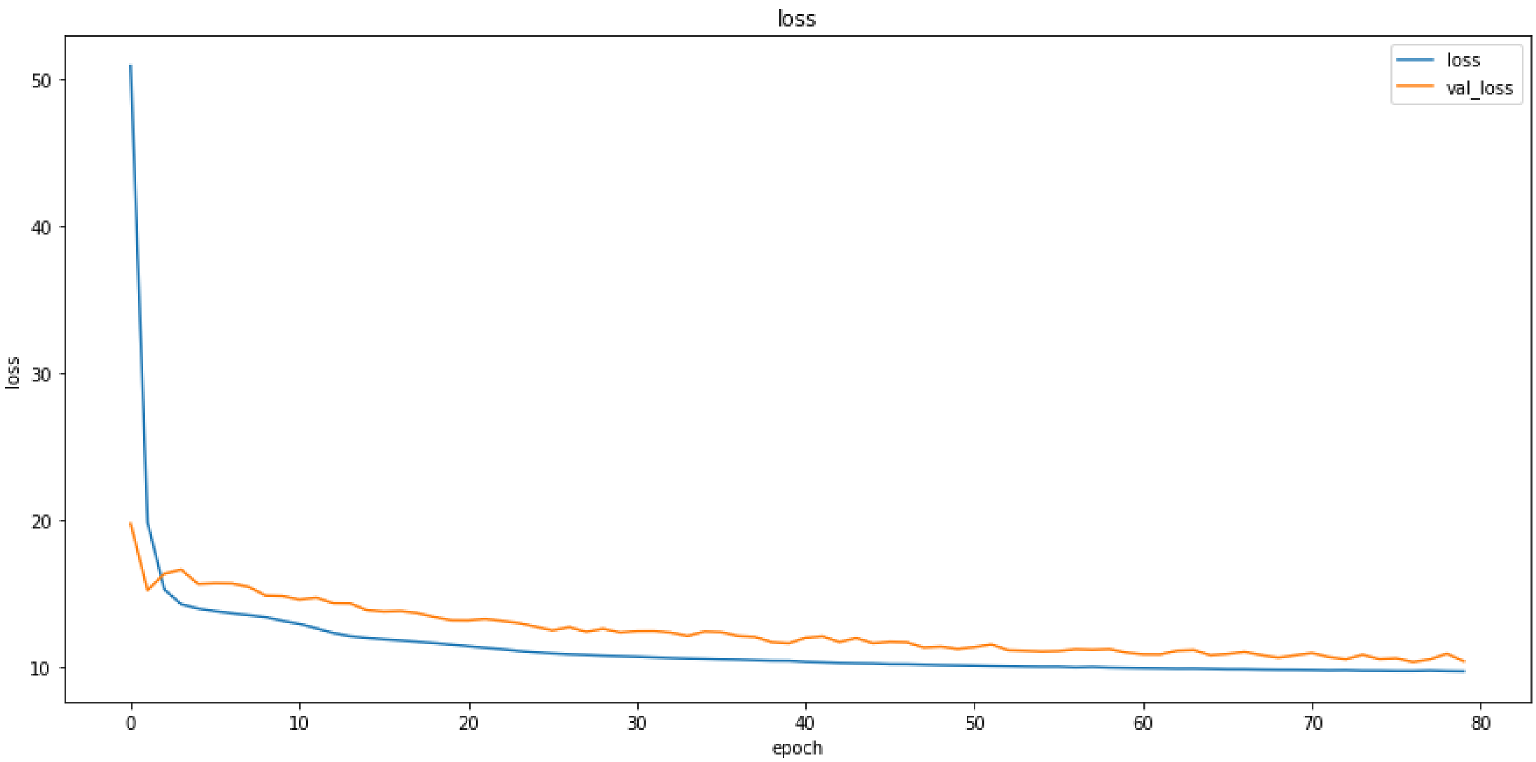


2.3 Report histogram plot, mean and std. dev. of normal data, and confusion matrix for 2 standard deviations as results. Discuss your loss plot. [+10 for greater than 75 TP, +10 all else]

Since the model performance in problem2.2 is good, I design a model based on the Convolutional AE in problem2.2 and further increase the number of convolutional kernels to faced with the larger input. The details of this model are shown in the below figure. For the encoder part, this model has three conv2d layers and three max-pooling layers, for the decoder part, this model has three conv2d layers and three up_sampling layers. For the number of convolutional kernels in each conv2d layer, I used to try **16-8-8-16, 32-16-16-32, 32-4-4-32 and 32-16-4-4-16-32**. Finally, I choose **32-16-4-4-16-32** since this one has the best performance and when the models are initialized randomly, this one can always get a stable result. Also, the overfitting problem is not very obvious for this combination, so I think the 32-16-4-4-16-32 has a good balance of complexity and performance.

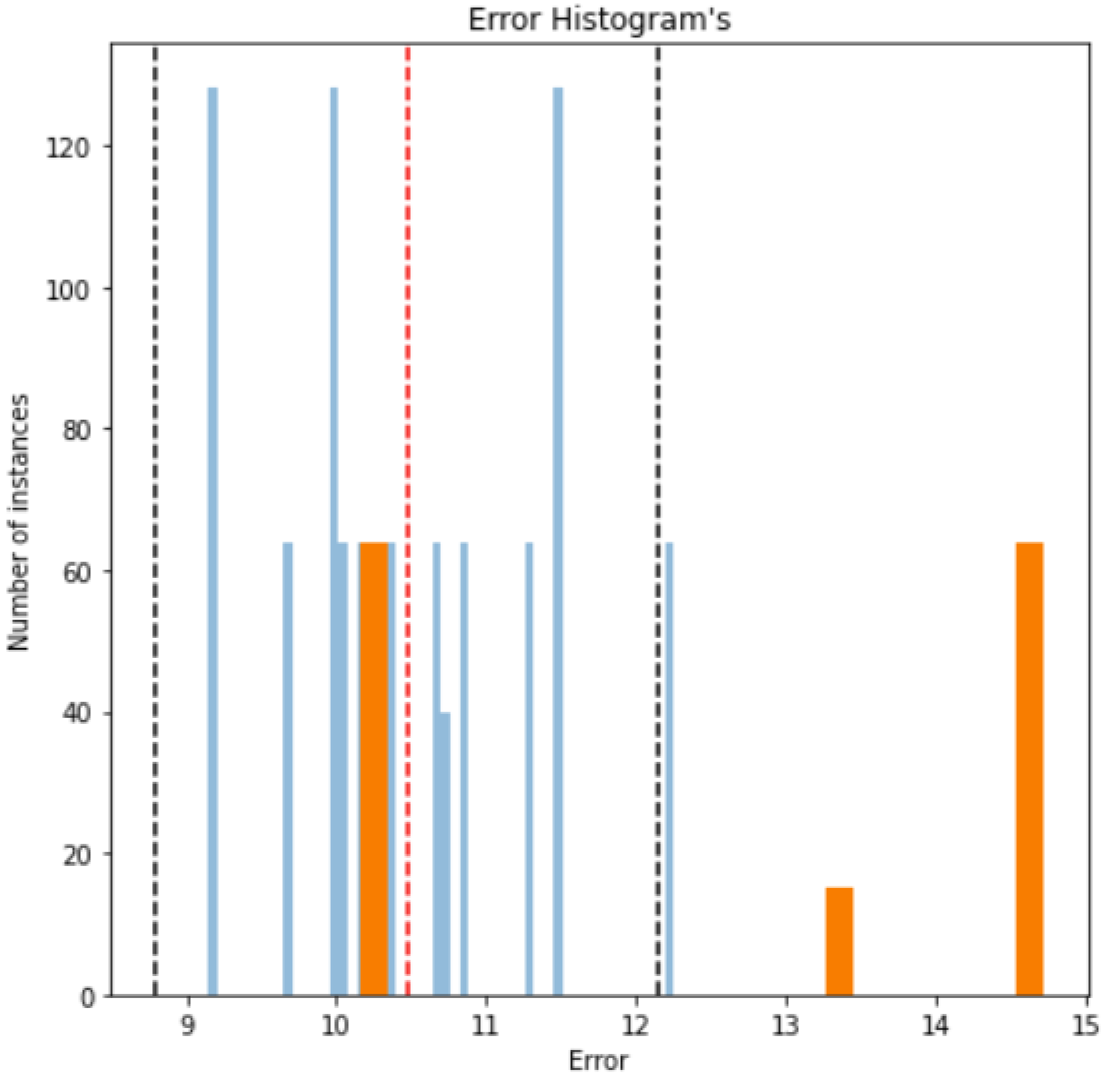
```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_131 (Conv2D)          (None, 64, 312, 32)       320
_________________________________________________________________
max_pooling2d_55 (MaxPooling (None, 32, 156, 32)       0
_________________________________________________________________
dropout_6 (Dropout)          (None, 32, 156, 32)       0
_________________________________________________________________
conv2d_132 (Conv2D)          (None, 32, 156, 16)       4624
_________________________________________________________________
max_pooling2d_56 (MaxPooling (None, 16, 78, 16)        0
_________________________________________________________________
conv2d_133 (Conv2D)          (None, 16, 78, 4)         580
_________________________________________________________________
max_pooling2d_57 (MaxPooling (None, 8, 39, 4)          0
_________________________________________________________________
conv2d_134 (Conv2D)          (None, 8, 39, 4)          148
_________________________________________________________________
up_sampling2d_53 (UpSampling (None, 16, 78, 4)         0
_________________________________________________________________
conv2d_135 (Conv2D)          (None, 16, 78, 16)        592
_________________________________________________________________
up_sampling2d_54 (UpSampling (None, 32, 156, 16)       0
_________________________________________________________________
conv2d_136 (Conv2D)          (None, 32, 156, 32)       4640
_________________________________________________________________
up_sampling2d_55 (UpSampling (None, 64, 312, 32)       0
_________________________________________________________________
conv2d_137 (Conv2D)          (None, 64, 312, 1)        289
=================================================================
Total params: 11,193
Trainable params: 11,193
Non-trainable params: 0
_________________________________________________________________
None
```

The overfitting problem for this model is not very secrous, so I just add one dropout layer and then we can see two very similar loss curves from the figure below. **I think the reason for the less overfitting is that convolutional layers just have few parameters, they need less regularization to begin with. Furthermore, the max-pooling layers can also take the relos of reducing the complexity of the model.**



To justify my model beyond the result: I ran it **repeatedly with random initializations for five times** and still receive the **same result(TP=79)**. So I do not think I get a lucky initialization in the weight space and the results are very stable. For the non-linearity I also try the sigmoid but I didn't get any good results based on that. I think the sigmoid function may be more useful in classification problems since the output is scaled nonlinearly but it may not be a good choice for image reconstruction. I trained my model for 150 and 80 epochs and I think **80 epochs** is better due to the similar performance and better time consumption. For anything in the data that affected model building, I think the most important thing is the **shape of the input data**. It can not only affect the input layers' shape but also affect how many max-pooling layers we can use. I eventually just used **one channel** from the input data because I noticed that one channel is more effective than others. I also try to combine eight channels data together and make an input data set like (1000,64,3138,1), but the results did not make any sense, so I just use one channel. **I think that means some time wash the data before the training process can significantly benefit the results.**

The mean of normal data is 10.4711 and standard deviation is 0.8402



Reminder: Achieve better than 75 anomalies

In [146]:
```
#Import dataset and normalize to [0,1]
mnist = tf.keras.datasets.mnist
(data_train, labels_train), (data_test, labels_test) = mnist.load_data()
#Normalize
data_train = data_train / 255
data_test = data_test /255
#Reshape
data_train = data_train.reshape(data_train.shape[0], 28, 28, 1)
data_test = data_test.reshape(data_test.shape[0], 28, 28, 1)

#Create labels as one-hot vectors
labels_train = tf.keras.utils.to_categorical(labels_train, num_classes=10)
labels_test = tf.keras.utils.to_categorical(labels_test, num_classes=10)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
```

Section 1 - CNN's

Fill in the model:

- Input: 28x28x1 grayscale image (1 specifies single channel grayscale).
- 1st hidden: 2D convolutional layer with 256 feature maps and 3x3 filters.
- 2nd hidden: A 2x2 maxpool layer.
- 3rd hidden: 2D convolutional layer with 128 feature maps and 3x3 filters.
- 4th hidden: A 2x2 maxpool layer.
- 5th hidden: Flatten layer to map 2D to 1D vector.
- 6th hidden: Dense layer of 100 perceptrons.
- 7th hidden: Dense layer of 100 perceptrons.
- Output: 10 perceptrons for classification.
  Activations, bias, loss function, and optimizer are your choice.
  Train for 200 epochs

1.1 Overfitting

In [147]:

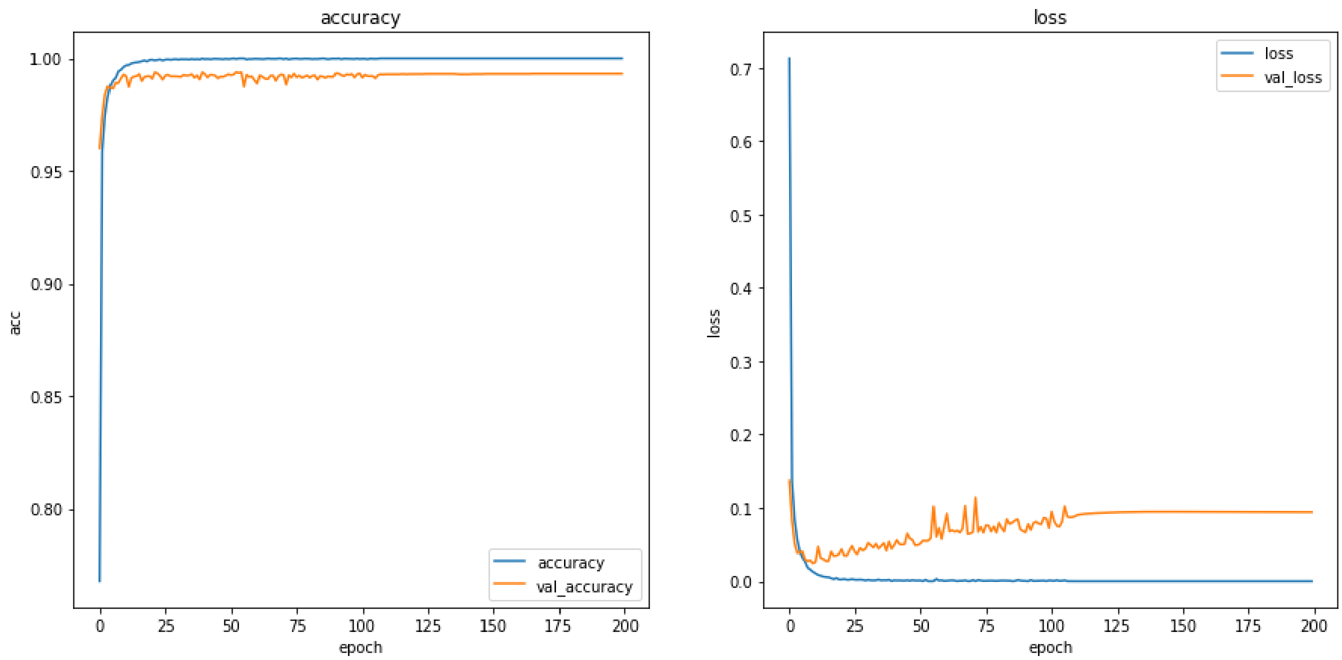
```
#Create and train model architecture
def CNN_overfit():

    #Easiest way to build model in Keras is using Squential. It allows model to be build layer by
 layer as we will do here
    model = tf.keras.models.Sequential()

    #### Fill in Model ####
    model.add(tf.keras.layers.Conv2D(256, kernel_size=(3, 3),activation='relu', input_shape=(28, 2
8, 1)))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Conv2D(128 , (3, 3), activation='relu'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))

    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(100, activation='relu'))
    model.add(tf.keras.layers.Dense(100, activation='relu'))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))


    return model

CNN_overfit = CNN_overfit()
print(CNN_overfit.summary())
CNN_overfit.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
history_overfit = CNN_overfit.fit(data_train, labels_train, validation_data=(data_test, labels_tes
t), epochs=10, batch_size=1000, shuffle=True)
scores = CNN_overfit.evaluate(data_test, labels_test)
print("Accuracy: %.2f%%" %(scores[1]*100))
```

```
Model: "sequential_25"
_________________________________________________________________
Layer (type)                Output Shape              Param #
=================================================================
conv2d_140 (Conv2D)         (None, 26, 26, 256)       2560
_________________________________________________________________
max_pooling2d_60 (MaxPooling (None, 13, 13, 256)      0
_________________________________________________________________
conv2d_141 (Conv2D)         (None, 11, 11, 128)       295040
_________________________________________________________________
max_pooling2d_61 (MaxPooling (None, 5, 5, 128)        0
_________________________________________________________________
flatten_1 (Flatten)         (None, 3200)              0
_________________________________________________________________
dense_3 (Dense)             (None, 100)               320100
_________________________________________________________________
dense_4 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_5 (Dense)             (None, 10)                1010
=================================================================
Total params: 628,810
Trainable params: 628,810
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
60/60 [==============================] - 12s 205ms/step - loss: 0.6882 - accuracy: 0.7825 - val_loss: 0.3163 - val_accuracy: 0.8999
Epoch 2/10
60/60 [==============================] - 12s 204ms/step - loss: 0.1432 - accuracy: 0.9552 - val_loss: 0.0772 - val_accuracy: 0.9752
Epoch 3/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0811 - accuracy: 0.9746 - val_loss: 0.0732 - val_accuracy: 0.9759
Epoch 4/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0562 - accuracy: 0.9829 - val_loss: 0.0359 - val_accuracy: 0.9887
Epoch 5/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0402 - accuracy: 0.9876 - val_loss: 0.0318 - val_accuracy: 0.9894
Epoch 6/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0316 - accuracy: 0.9899 - val_loss: 0.0478 - val_accuracy: 0.9838
Epoch 7/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0259 - accuracy: 0.9919 - val_loss: 0.0273 - val_accuracy: 0.9918
Epoch 8/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0199 - accuracy: 0.9937 - val_loss: 0.0238 - val_accuracy: 0.9924
Epoch 9/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0162 - accuracy: 0.9950 - val_loss: 0.0269 - val_accuracy: 0.9913
Epoch 10/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0126 - accuracy: 0.9960 - val_loss: 0.0276 - val_accuracy: 0.9912
313/313 [==============================] - 2s 5ms/step - loss: 0.0276 - accuracy: 0.9912
Accuracy: 99.12%
```

In [0]:
```python
# Information contained in history dict.
print(history_overfit.history.keys())
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [0]:
```python
#acc scores = 99.32%
#Plot accuracy vs epoch
plt.figure(figsize=(15,7))
plt.subplot(121)
plt.plot(history_overfit.history['accuracy'], label='accuracy')
plt.plot(history_overfit.history['val_accuracy'], label='val_accuracy')
plt.legend(loc='lower right')
plt.title('accuracy')
plt.xlabel('epoch')
plt.ylabel('acc')
#### Fill in plot ####

#Plot loss vs epoch
plt.subplot(122)
plt.plot(history_overfit.history['loss'],label='loss')
plt.plot(history_overfit.history['val_loss'],label='val_loss')
plt.legend(loc='upper right')
plt.title('loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
#### Fill in plot ####
```



1.2 Improvements

Using the network above, (1) insert a dropout of 30% between the input and first hidden layer. Run the model again and make note of the result. Next, (2) remove the dropout between input and hidden and add a dropout to each hidden layer except between softmax and output layer. Plot accuracy and loss only for (2). What do you observe for (2)?.

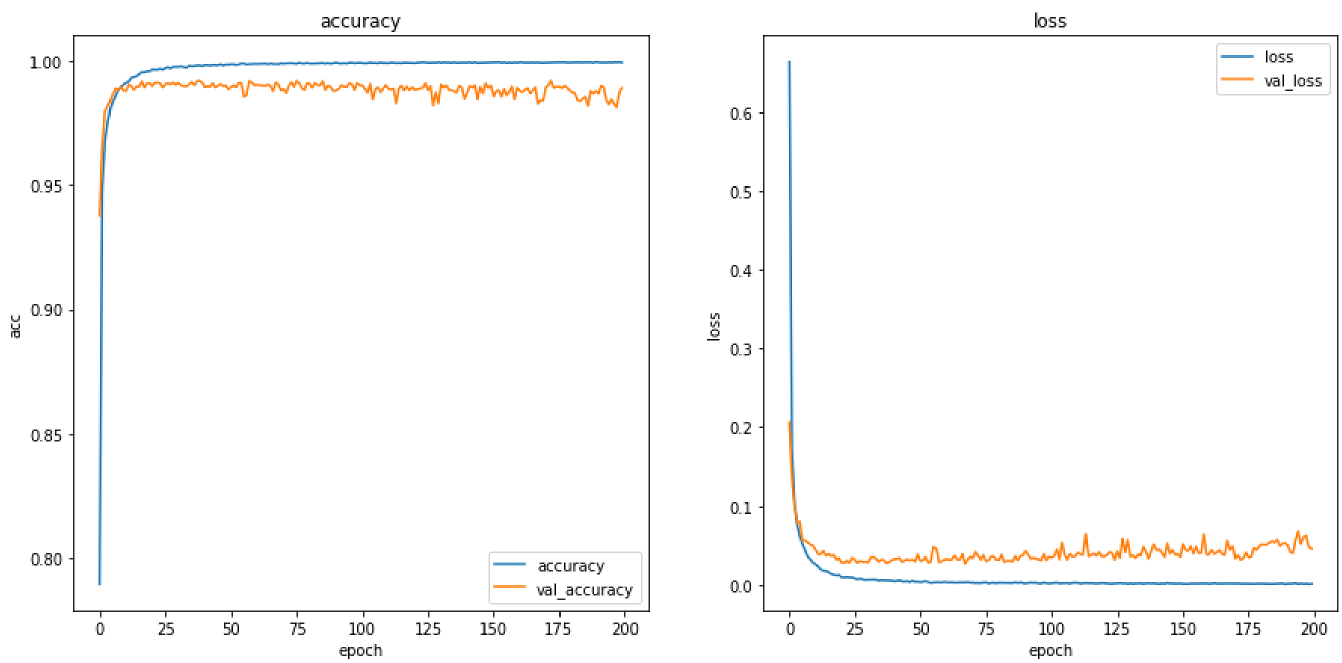
For 1.2 (1):

In [148]:
```
def CNN_dropout_hidden():
    model = tf.keras.models.Sequential()
    #### Fill in model ####
    model.add(tf.keras.layers.Input(shape=(28, 28, 1)))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Conv2D(256, kernel_size=(3, 3),activation='relu'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Conv2D(128 , (3, 3), activation='relu'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dense(100, activation='relu'))
    model.add(tf.keras.layers.Dense(100, activation='relu'))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    return model

#Compile and train the model
CNN_dropout_hidden = CNN_dropout_hidden()
CNN_dropout_hidden.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accurac
y'])
history_dropout_hidden = CNN_dropout_hidden.fit(data_train, labels_train, validation_data=(data_te
st, labels_test), epochs=10, batch_size=1000, shuffle=True)
scores_dropout_hidden = CNN_dropout_hidden.evaluate(data_test, labels_test)
print("Accuracy: %.2f%%" %(scores_dropout_hidden[1]*100))
```

```
Epoch 1/10
60/60 [==============================] - 12s 205ms/step - loss: 0.7257 - accuracy: 0.7597 - val_
loss: 0.2435 - val_accuracy: 0.9432
Epoch 2/10
60/60 [==============================] - 12s 204ms/step - loss: 0.1565 - accuracy: 0.9508 - val_
loss: 0.1395 - val_accuracy: 0.9606
Epoch 3/10
60/60 [==============================] - 12s 204ms/step - loss: 0.1022 - accuracy: 0.9673 - val_
loss: 0.0919 - val_accuracy: 0.9771
Epoch 4/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0748 - accuracy: 0.9766 - val_
loss: 0.0908 - val_accuracy: 0.9830
Epoch 5/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0584 - accuracy: 0.9821 - val_
loss: 0.0690 - val_accuracy: 0.9840
Epoch 6/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0488 - accuracy: 0.9850 - val_
loss: 0.0657 - val_accuracy: 0.9875
Epoch 7/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0411 - accuracy: 0.9865 - val_
loss: 0.0827 - val_accuracy: 0.9871
Epoch 8/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0350 - accuracy: 0.9889 - val_
loss: 0.0577 - val_accuracy: 0.9867
Epoch 9/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0300 - accuracy: 0.9899 - val_
loss: 0.0394 - val_accuracy: 0.9905
Epoch 10/10
60/60 [==============================] - 12s 204ms/step - loss: 0.0279 - accuracy: 0.9907 - val_
loss: 0.0563 - val_accuracy: 0.9878
313/313 [==============================] - 2s 5ms/step - loss: 0.0563 - accuracy: 0.9878
Accuracy: 98.78%
```

In [0]:

```
# acc scores = 98.92%
# Plot accuracy vs epoch
plt.figure(figsize=(15,7))
plt.subplot(121)
plt.plot(history_dropout_hidden.history['accuracy'], label='accuracy')
plt.plot(history_dropout_hidden.history['val_accuracy'], label='val_accuracy')
plt.legend(loc='lower right')
plt.title('accuracy')
plt.xlabel('epoch')
plt.ylabel('acc')
#### Fill in plot ####

#Plot loss vs epoch
plt.subplot(122)
plt.plot(history_dropout_hidden.history['loss'],label='loss')
plt.plot(history_dropout_hidden.history['val_loss'],label='val_loss')
plt.legend(loc='upper right')
plt.title('loss')
plt.xlabel('epoch')
plt.ylabel('loss')
#### Fill in plot ####
plt.show()
```



For 1.2 (2):

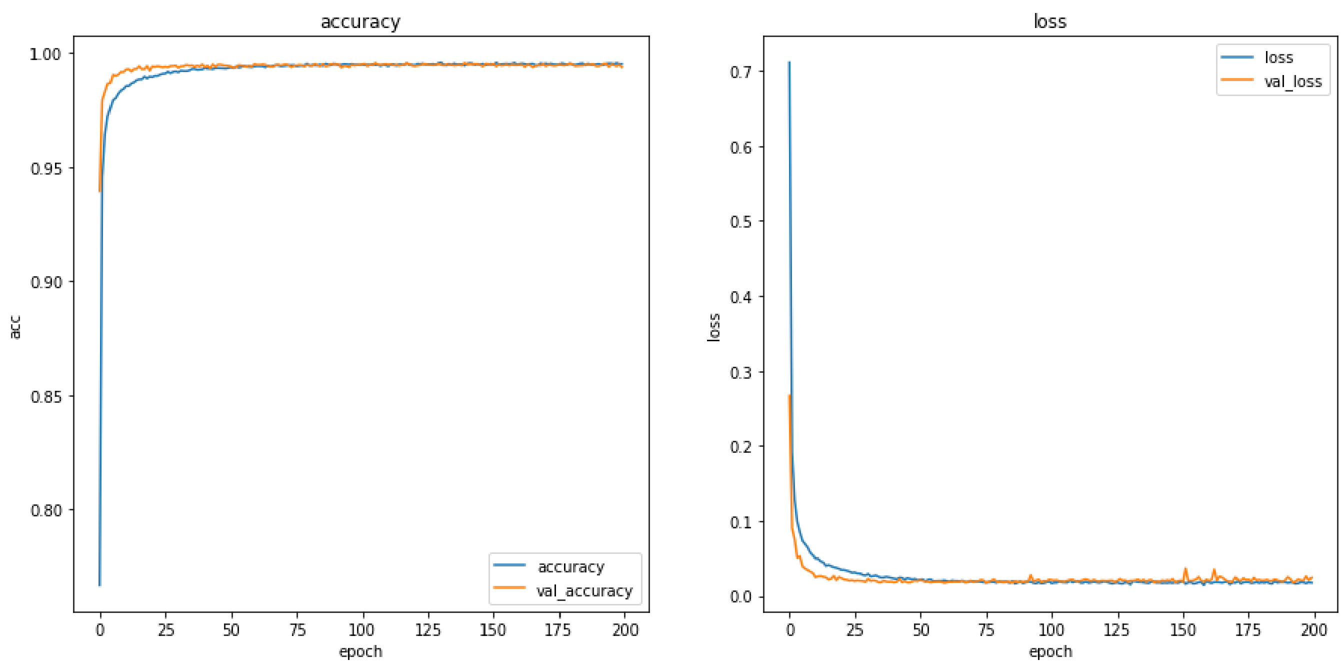
In [149]:

```
#Create and train model architecture
def CNN_dropout_hidden2():
    model = tf.keras.models.Sequential()
    #### Fill in model ####
    model.add(tf.keras.layers.Conv2D(256, kernel_size=(3, 3),activation='relu', input_shape=(28, 2
8, 1)))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Conv2D(128 , (3, 3), activation='relu'))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Flatten())
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(100, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(100, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(10, activation='softmax'))

    return model

#Compile and train the model
CNN_dropout_hidden2 = CNN_dropout_hidden2()
CNN_dropout_hidden2.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accura
cy'])
history_dropout_hidden2 = CNN_dropout_hidden2.fit(data_train, labels_train, validation_data=(data_
test, labels_test), epochs=10, batch_size=1000, shuffle=True)
scores_dropout_hidden2 = CNN_dropout_hidden2.evaluate(data_test, labels_test)
print("Accuracy: %.2f%%" %(scores_dropout_hidden2[1]*100))
```

```
Epoch 1/10
60/60 [==============================] - 15s 258ms/step - loss: 0.7937 - accuracy: 0.7398 - val_
loss: 0.2112 - val_accuracy: 0.9544
Epoch 2/10
60/60 [==============================] - 15s 256ms/step - loss: 0.2062 - accuracy: 0.9398 - val_
loss: 0.1037 - val_accuracy: 0.9762
Epoch 3/10
60/60 [==============================] - 15s 256ms/step - loss: 0.1370 - accuracy: 0.9596 - val_
loss: 0.0825 - val_accuracy: 0.9800
Epoch 4/10
60/60 [==============================] - 15s 256ms/step - loss: 0.1051 - accuracy: 0.9702 - val_
loss: 0.0549 - val_accuracy: 0.9872
Epoch 5/10
60/60 [==============================] - 15s 256ms/step - loss: 0.0888 - accuracy: 0.9744 - val_
loss: 0.0538 - val_accuracy: 0.9881
Epoch 6/10
60/60 [==============================] - 15s 256ms/step - loss: 0.0766 - accuracy: 0.9777 - val_
loss: 0.0442 - val_accuracy: 0.9883
Epoch 7/10
60/60 [==============================] - 15s 256ms/step - loss: 0.0663 - accuracy: 0.9804 - val_
loss: 0.0376 - val_accuracy: 0.9909
Epoch 8/10
60/60 [==============================] - 15s 256ms/step - loss: 0.0611 - accuracy: 0.9822 - val_
loss: 0.0338 - val_accuracy: 0.9915
Epoch 9/10
60/60 [==============================] - 15s 256ms/step - loss: 0.0578 - accuracy: 0.9834 - val_
loss: 0.0310 - val_accuracy: 0.9926
Epoch 10/10
60/60 [==============================] - 15s 256ms/step - loss: 0.0561 - accuracy: 0.9839 - val_
loss: 0.0300 - val_accuracy: 0.9924
313/313 [==============================] - 2s 5ms/step - loss: 0.0300 - accuracy: 0.9924
Accuracy: 99.24%
```

In [0]:

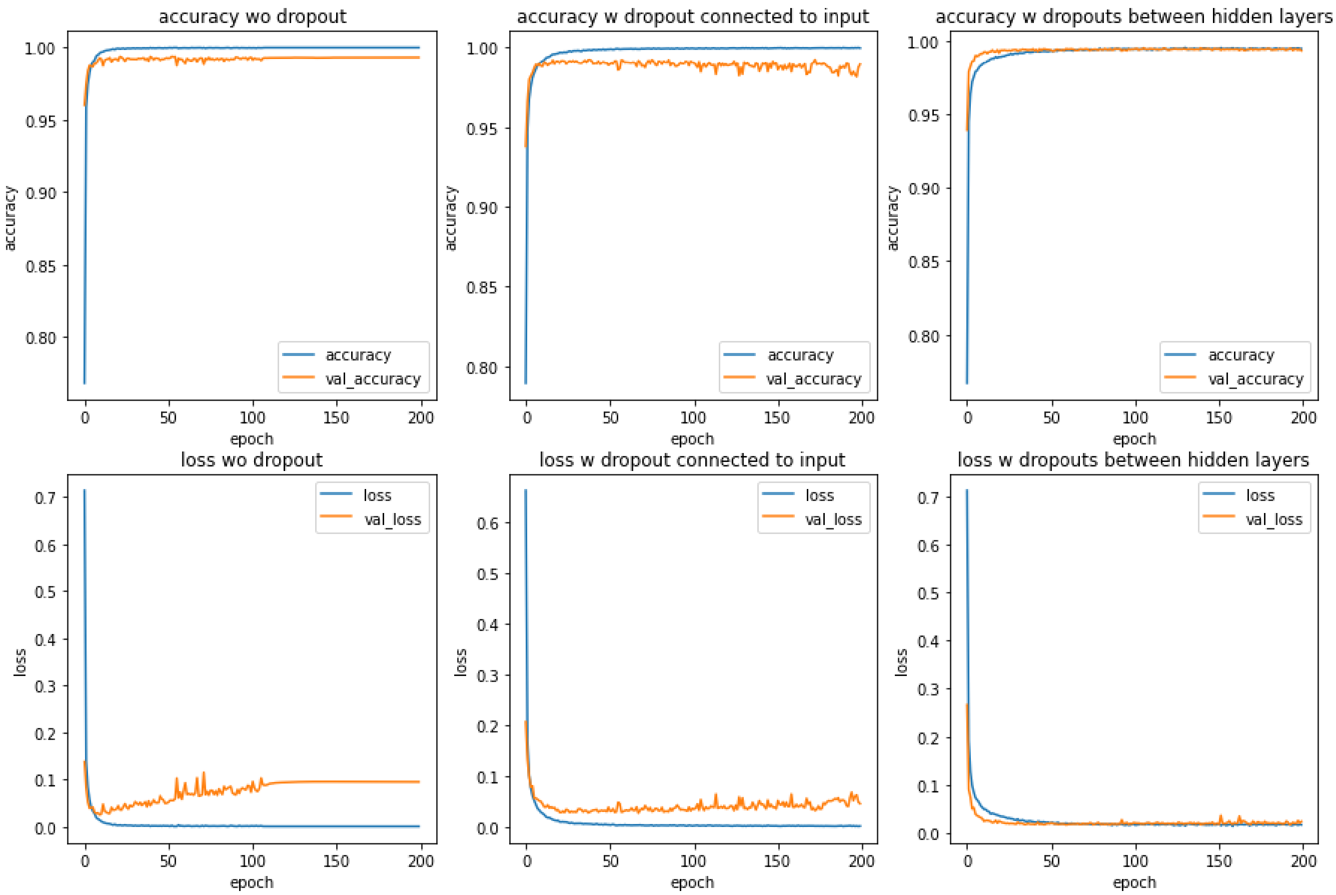
```
# acc scores = 99.37%
# Plot accuracy vs epoch
plt.figure(figsize=(15,7))
plt.subplot(121)
plt.plot(history_dropout_hidden2.history['accuracy'], label='accuracy')
plt.plot(history_dropout_hidden2.history['val_accuracy'], label='val_accuracy')
plt.legend(loc='lower right')
plt.title('accuracy')
plt.xlabel('epoch')
plt.ylabel('acc')
#### Fill in plot ####

#Plot loss vs epoch
plt.subplot(122)
plt.plot(history_dropout_hidden2.history['loss'],label='loss')
plt.plot(history_dropout_hidden2.history['val_loss'],label='val_loss')
plt.legend(loc='upper right')
plt.title('loss')
plt.xlabel('epoch')
plt.ylabel('loss')
#### Fill in plot ####
plt.show()
```



Summary for 1.2 Dropout improvements

In [0]:

```
plt.figure(figsize=(15,10))

# Plot accuracy vs epoch
plt.subplot(231)
plt.plot(history_overfit.history['accuracy'], label='accuracy')
plt.plot(history_overfit.history['val_accuracy'], label='val_accuracy')
plt.legend(loc='lower right')
plt.title('accuracy wo dropout')
plt.xlabel('epoch')
plt.ylabel('accuracy')

plt.subplot(232)
plt.plot(history_dropout_hidden.history['accuracy'],label='accuracy')
plt.plot(history_dropout_hidden.history['val_accuracy'],label='val_accuracy')
plt.legend(loc='lower right')
plt.title('accuracy w dropout connected to input')
plt.xlabel('epoch')
plt.ylabel('accuracy')

plt.subplot(233)
plt.plot(history_dropout_hidden2.history['accuracy'],label='accuracy')
plt.plot(history_dropout_hidden2.history['val_accuracy'],label='val_accuracy')
plt.legend(loc='lower right')
plt.title('accuracy w dropouts between hidden layers')
plt.xlabel('epoch')
plt.ylabel('accuracy')
#### Fill in plot ####

#Plot loss vs epoch
plt.subplot(234)
plt.plot(history_overfit.history['loss'], label='loss')
plt.plot(history_overfit.history['val_loss'], label='val_loss')
plt.legend(loc='upper right')
plt.title('loss wo dropout')
plt.xlabel('epoch')
plt.ylabel('loss')

plt.subplot(235)
plt.plot(history_dropout_hidden.history['loss'],label='loss')
plt.plot(history_dropout_hidden.history['val_loss'],label='val_loss')
plt.legend(loc='upper right')
plt.title('loss w dropout connected to input')
plt.xlabel('epoch')
plt.ylabel('loss')

plt.subplot(236)
plt.plot(history_dropout_hidden2.history['loss'],label='loss')
plt.plot(history_dropout_hidden2.history['val_loss'],label='val_loss')
plt.legend(loc='upper right')
plt.title('loss w dropouts between hidden layers')
plt.xlabel('epoch')
plt.ylabel('loss')
#### Fill in plot ####
plt.show()
```



Section 2- Autoencoders

2.1 Linear AE

Fill in the model:

- Input: Flattened grayscale image to 28^2 = 784-dimensional vector.
- 1st hidden: 400 perceptrons.
- 2nd hidden: 200 perceptrons.
- 3rd hidden: 100 perceptrons.
- 4th hidden: 200 perceptrons.
- 5th hidden: 400 perceptrons.
- Output: 784 perceptrons.

  Train for 150 epochs

In [0]:

```
#Reshape training and testing data
data_train_reshape_fcae = data_train.reshape(data_train.shape[0], 784)
data_test_reshape_fcae = data_test.reshape(data_test.shape[0], 784)
```

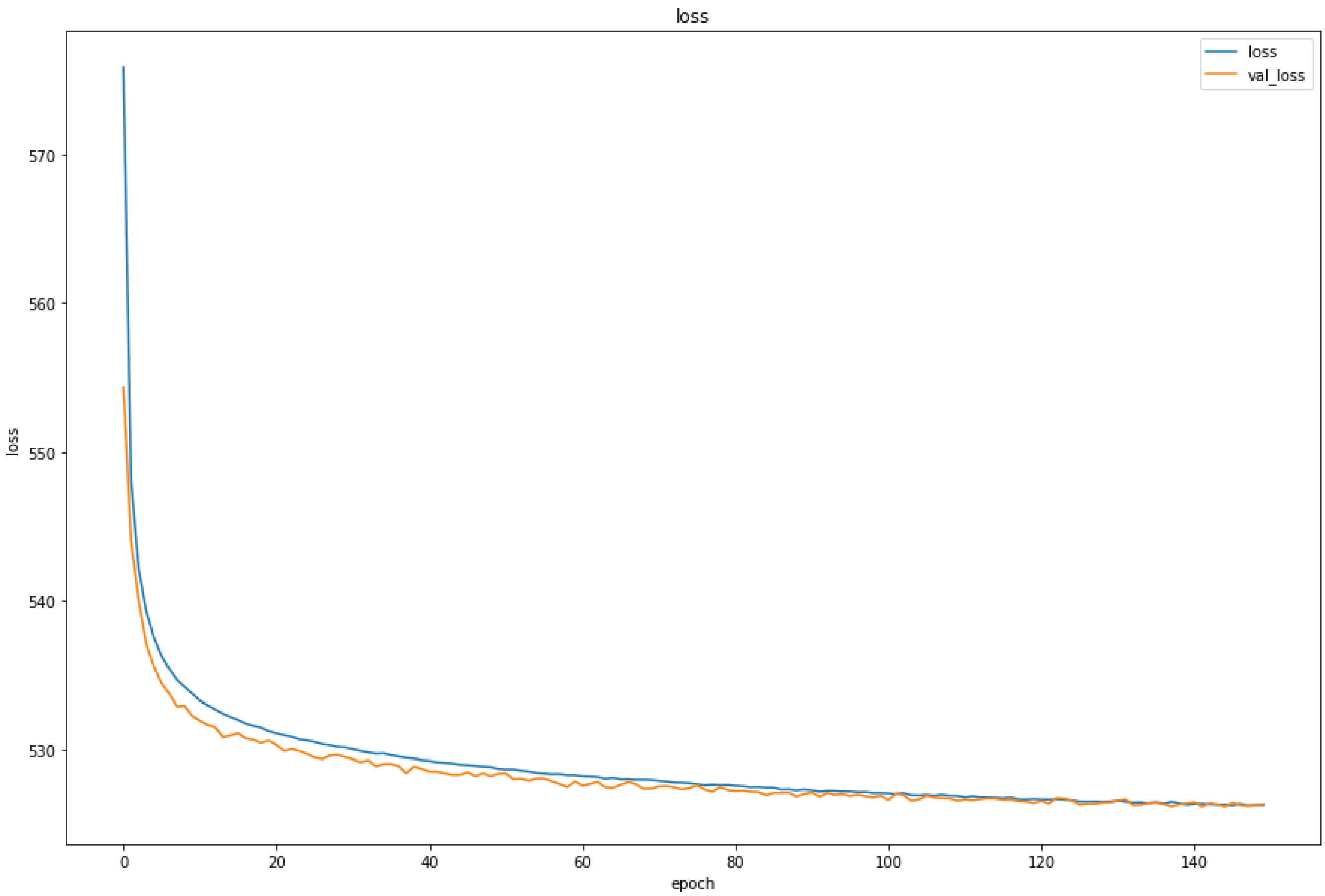
In [154]:

```
# Create autoencoder architecture
def deep_ae():
    model = tf.keras.models.Sequential()

    model.add(tf.keras.layers.Input(shape=(784,)))
    # Encoder

    #### Fill in the model ####
    model.add(tf.keras.layers.Dense(400, activation='relu'))
    #model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(200, activation='relu'))
    #model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(100, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.3))
    # Decoder

    #### Fill in the model ####
    model.add(tf.keras.layers.Dense(200, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(400, activation='relu'))
    model.add(tf.keras.layers.Dropout(0.3))
    model.add(tf.keras.layers.Dense(784, activation='sigmoid'))

    return model

#Create deep autoencoder graph, compile it to use mean squared error loss and the adam optimizer,
 train the model, create predictions
deep_ae = deep_ae()
print(deep_ae.summary())
deep_ae.compile(loss='categorical_crossentropy', optimizer='adam')
history_deep_ae = deep_ae.fit(data_train_reshape_fcae, data_train_reshape_fcae, validation_data=(d
ata_test_reshape_fcae, data_test_reshape_fcae), epochs=10, batch_size=250, shuffle=True)
decoded_data = deep_ae.predict(data_test_reshape_fcae)

#Obtain encoder representation of data
get_hl = K.function([deep_ae.layers[0].input], [deep_ae.layers[2].output])
deep_ae_hl = get_hl([data_test_reshape_fcae])[0]
```

```
Model: "sequential_30"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_24 (Dense)             (None, 400)               314000
_________________________________________________________________
dense_25 (Dense)             (None, 200)               80200
_________________________________________________________________
dense_26 (Dense)             (None, 100)               20100
_________________________________________________________________
dropout_21 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_27 (Dense)             (None, 200)               20200
_________________________________________________________________
dropout_22 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_28 (Dense)             (None, 400)               80400
_________________________________________________________________
dropout_23 (Dropout)         (None, 400)               0
_________________________________________________________________
dense_29 (Dense)             (None, 784)               314384
=================================================================
Total params: 829,284
Trainable params: 829,284
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
240/240 [==============================] - 1s 4ms/step - loss: 575.7485 - val_loss: 554.7593
Epoch 2/10
240/240 [==============================] - 1s 4ms/step - loss: 548.4683 - val_loss: 544.8527
Epoch 3/10
240/240 [==============================] - 1s 4ms/step - loss: 542.5052 - val_loss: 539.8880
Epoch 4/10
240/240 [==============================] - 1s 4ms/step - loss: 539.5171 - val_loss: 537.9699
Epoch 5/10
240/240 [==============================] - 1s 4ms/step - loss: 537.7516 - val_loss: 535.8461
Epoch 6/10
240/240 [==============================] - 1s 4ms/step - loss: 536.5424 - val_loss: 534.6545
Epoch 7/10
240/240 [==============================] - 1s 4ms/step - loss: 535.5967 - val_loss: 533.7915
Epoch 8/10
240/240 [==============================] - 1s 4ms/step - loss: 534.9063 - val_loss: 533.5900
Epoch 9/10
240/240 [==============================] - 1s 4ms/step - loss: 534.4302 - val_loss: 532.9552
Epoch 10/10
240/240 [==============================] - 1s 4ms/step - loss: 533.9613 - val_loss: 532.4114
```

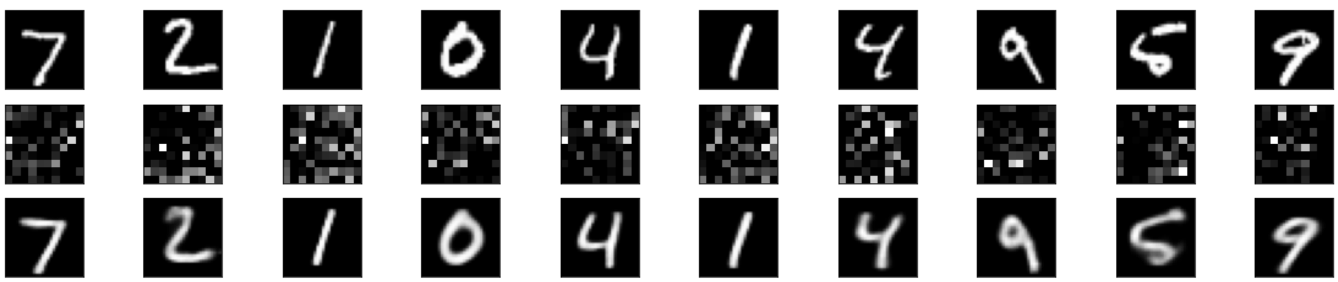
In [0]:

```
#Plot train/validation loss vs epoch
plt.figure(figsize=(15,10))
#Plot loss vs epoch
plt.plot(history_deep_ae.history['loss'],label='loss')
plt.plot(history_deep_ae.history['val_loss'],label='val_loss')
plt.legend(loc='upper right')
plt.title('loss')
plt.xlabel('epoch')
plt.ylabel('loss')
#### Fill in the plot ####
plt.show()
```



In [0]:

```
#Plot samples of 10 images, their hidden layer representations, and their reconstructions
n = 10  # how many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(data_test_reshape_fcae[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display hidden layer representation
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(deep_ae_hl[i].reshape(10, 10))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(3, n, i + 1 + n + n)
    plt.imshow(decoded_data[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



2.2 Convolutional AE

Fill in the model:

- Input: 28x28x1 grayscale image.
- 1st hidden: 2D convolutional layer with 16 feature maps and 3x3 filters.
- 2nd hidden: A 2x2 maxpool layer.
- 3rd hidden: 2D convolutional layer with 8 feature maps and 3x3 filters.
- 4th hidden: A 2x2 maxpool layer.
- 5th hidden: 2D convolutional layer with 8 feature maps and 3x3 filters.
- 6th hidden: A 2x2 upsample layer.
- 7th hidden: 2D convolutional layer with 16 feature maps and 3x3 filters.
- 8th hidden: A 2x2 upsample layer.
- Output: A convolutional layer with a single feature map and 3x3 filters.
  All experiments with dropout set at 30%. Train for 200 epochs

In [0]:

```
#Reshape data to account for grayscale channel in each image
data_train_reshape_cae = data_train.reshape(data_train.shape[0], 28, 28, 1)
data_test_reshape_cae = data_test.reshape(data_test.shape[0], 28, 28, 1)
```

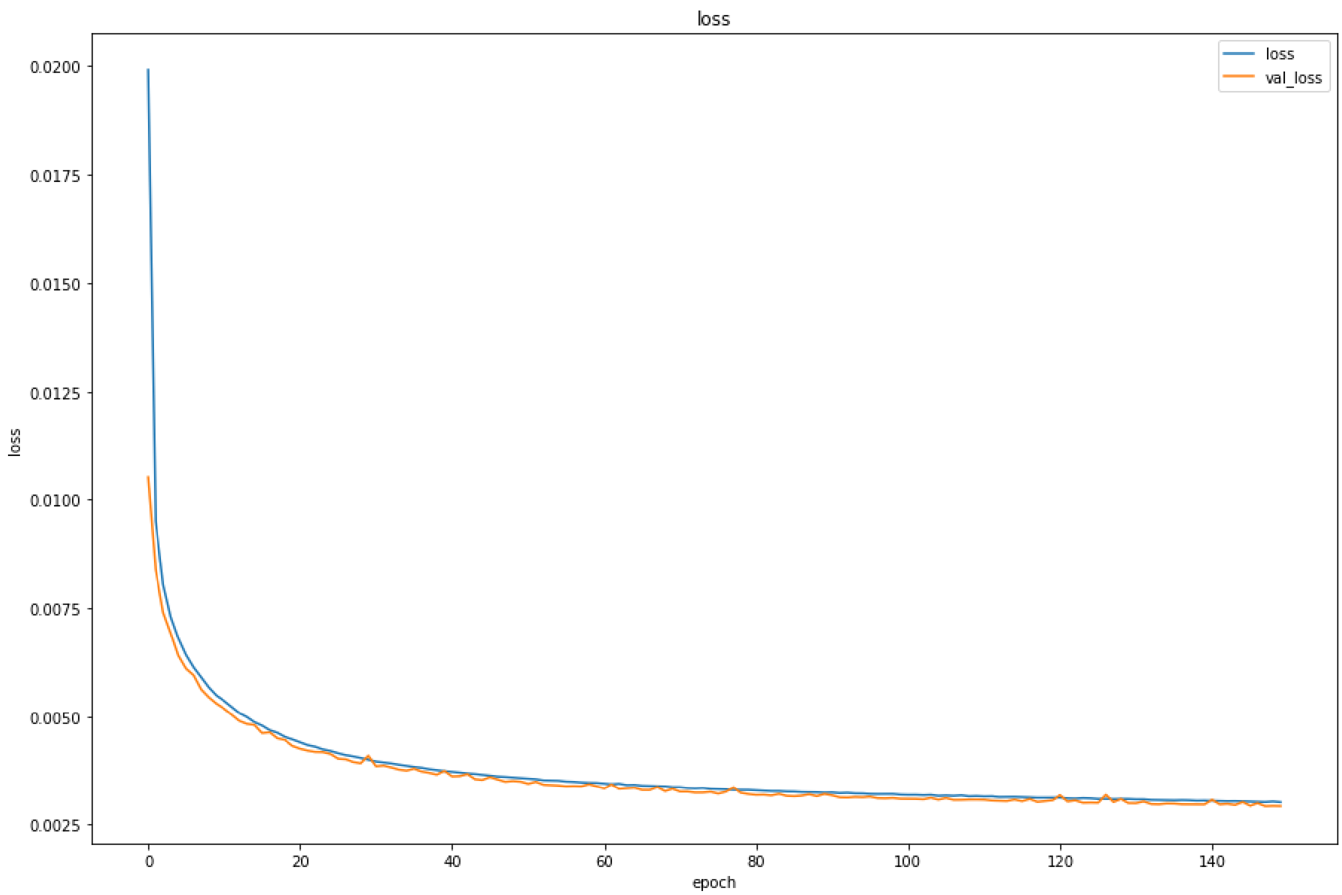
In [157]:
```
#Create Convolutional AutoEncoder Architecture
def cae():
    model = tf.keras.models.Sequential()

    #Encoder
    model.add(tf.keras.layers.Conv2D(16, kernel_size=(3, 3),activation='relu', padding='same', inp
ut_shape=(28, 28, 1)))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'))
    model.add(tf.keras.layers.Conv2D(8 , (3, 3), activation='relu',padding='same'))
    model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'))
    #### Fill in model ####

    #Decoder
    model.add(tf.keras.layers.Conv2D(8 , (3, 3), activation='relu',padding='same'))
    model.add(tf.keras.layers.UpSampling2D((2, 2)))
    model.add(tf.keras.layers.Conv2D(16 , (3, 3), activation='relu',padding='same'))
    model.add(tf.keras.layers.UpSampling2D((2, 2)))
    #### Fill in model ####
    model.add(tf.keras.layers.Conv2D(1 , (3, 3), activation='relu',padding='same'))

    return model

conv_ae = cae()
print(conv_ae.summary())

#Create deep autoencoder graph, compile it to use mean squared error loss and the adam optimizer,
 train the model, create predictions
conv_ae.compile(loss='mse', optimizer='adam')
history_conv_ae = conv_ae.fit(data_train_reshape_cae, data_train_reshape_cae, validation_data=(dat
a_test_reshape_cae, data_test_reshape_cae), epochs=10, batch_size=250, shuffle=True)
decoded_data = conv_ae.predict(data_test_reshape_cae)

#Obtain encoder representation of data
get_hl = K.function([conv_ae.layers[0].input], [conv_ae.layers[3].output])
conv_ae_hl = get_hl([data_test_reshape_cae])[0]
```

```
Model: "sequential_32"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_151 (Conv2D)          (None, 28, 28, 16)        160       
_________________________________________________________________
max_pooling2d_68 (MaxPooling (None, 14, 14, 16)        0         
_________________________________________________________________
conv2d_152 (Conv2D)          (None, 14, 14, 8)         1160      
_________________________________________________________________
max_pooling2d_69 (MaxPooling (None, 7, 7, 8)           0         
_________________________________________________________________
conv2d_153 (Conv2D)          (None, 7, 7, 8)           584       
_________________________________________________________________
up_sampling2d_58 (UpSampling (None, 14, 14, 8)         0         
_________________________________________________________________
conv2d_154 (Conv2D)          (None, 14, 14, 16)        1168      
_________________________________________________________________
up_sampling2d_59 (UpSampling (None, 28, 28, 16)        0         
_________________________________________________________________
conv2d_155 (Conv2D)          (None, 28, 28, 1)         145       
=================================================================
Total params: 3,217
Trainable params: 3,217
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
240/240 [==============================] - 2s 9ms/step - loss: 0.0254 - val_loss: 0.0102
Epoch 2/10
240/240 [==============================] - 2s 8ms/step - loss: 0.0089 - val_loss: 0.0077
Epoch 3/10
240/240 [==============================] - 2s 8ms/step - loss: 0.0074 - val_loss: 0.0067
Epoch 4/10
240/240 [==============================] - 2s 8ms/step - loss: 0.0066 - val_loss: 0.0061
Epoch 5/10
240/240 [==============================] - 2s 8ms/step - loss: 0.0060 - val_loss: 0.0056
Epoch 6/10
240/240 [==============================] - 2s 8ms/step - loss: 0.0056 - val_loss: 0.0054
Epoch 7/10
240/240 [==============================] - 2s 8ms/step - loss: 0.0053 - val_loss: 0.0051
Epoch 8/10
240/240 [==============================] - 2s 8ms/step - loss: 0.0051 - val_loss: 0.0049
Epoch 9/10
240/240 [==============================] - 2s 9ms/step - loss: 0.0049 - val_loss: 0.0047
Epoch 10/10
240/240 [==============================] - 2s 8ms/step - loss: 0.0048 - val_loss: 0.0046
```

In [0]:

```
#Plot train/validation loss vs epoch
#Plot loss vs epoch
plt.figure(figsize=(15,10))
plt.plot(history_conv_ae.history['loss'],label='loss')
plt.plot(history_conv_ae.history['val_loss'],label='val_loss')
plt.legend(loc='upper right')
plt.title('loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.plot()
#### Fill in the plot ####
```

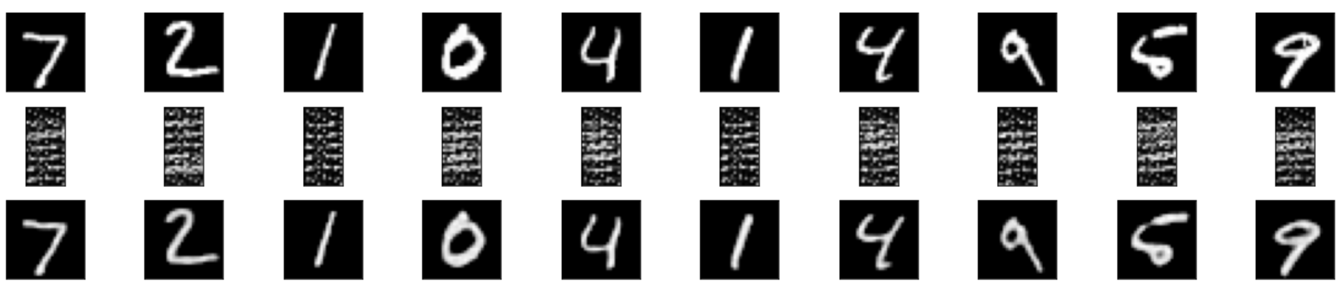
Out[0]: []



In [0]:

```python
#Plot samples of 10 images, their hidden layer representations, and their reconstructions
n = 10  # how many digits we will display
plt.figure(3)
plt.figure(figsize=(20, 4))
for i in range(n):
    # display original
    ax = plt.subplot(3, n, i + 1)
    plt.imshow(data_test_reshape_cae[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display hidden layer representation
    ax = plt.subplot(3, n, i + 1 + n)
    plt.imshow(conv_ae_hl[i].reshape(28, 14))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # display reconstruction
    ax = plt.subplot(3, n, i + 1 + n + n)
    plt.imshow(decoded_data[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



2.3 Machine Anomaly Detection

At this point you have enough starter code to Using the dataset provided **create the autoencdoer model** you deem necessary to achieve better than 75 true positives (TP = 75) where a true instance is an anomaly. Or detect all 143 if you can! Although anomaly detection thresholds can be set arbitrarily and various metrics are used depending on the problem, we will set ours at 2 standard deviations from the mean of "normal" data to judge TP's. Use the code provided at the bottom for calculating true positives and histogramming.

In [0]:
```
#### Restart your kernal and run from here to clear some memory
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow import keras
tf.keras.backend.set_floatx('float64')

import sys
from os import listdir
from os.path import isfile, join
```

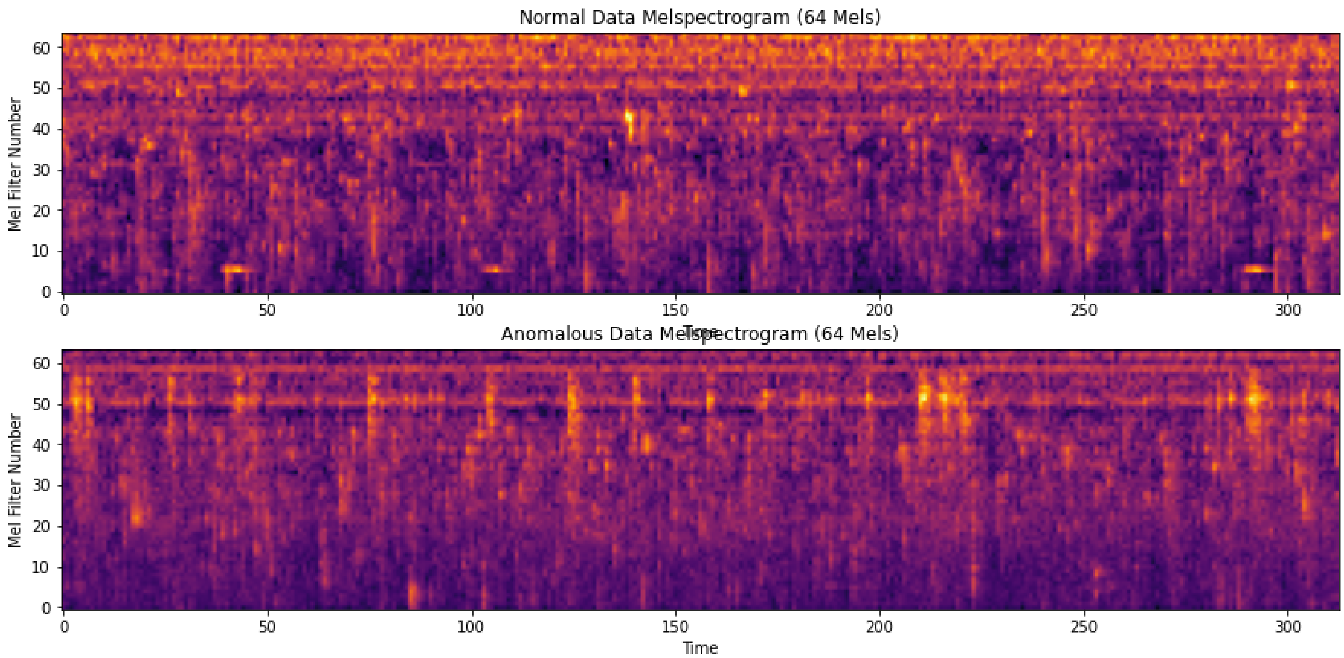
In [0]:
```
### I am using colab and the data is saved in my google drive so I need:
print('attention! using colab here and data is in my google drive!')
from google.colab import drive
drive.mount('/content/drive')
```

```
attention! using colab here and data is in my google drive!
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6b
n6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3
a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%
20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdr
ive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
datapath = './drive/My Drive/UCSD/2020/ECE228/PY2/'
```

Example spectrograms

In [0]:
```python
#### Load melspectrograms
ex_norm  = np.load(datapath + 'ex_normalspec.npy')
ex_anom = np.load(datapath + 'ex_abnormspec.npy')
plt.figure(figsize=(15,7))
plt.subplot(211)
plt.imshow(ex_norm[0,::-1], origin='lower', cmap='inferno')
plt.xlabel('Time')
plt.ylabel('Mel Filter Number')
plt.title('Normal Data Melspectrogram (64 Mels)')

plt.subplot(212)
plt.imshow(ex_anom[0,::-1], origin='lower', cmap='inferno')
plt.xlabel('Time')
plt.ylabel('Mel Filter Number')
plt.title('Anomalous Data Melspectrogram (64 Mels)')
plt.show()
```



General template, up to this point, for constructing your deep learning model

1. Set up the data (reshape, scale, etc...
2. Initialize a loss function
3. Compile a model
4. Train a model

In [142]:

```
# Create your own Baseline autoencoder
# Model name is fixed for use by later code
autoencoderBASE = tf.keras.models.Sequential([
  #### Fill in your model #####
  keras.layers.Input(shape=(64,312, 1)),

  keras.layers.Conv2D(32, kernel_size=(3, 3),activation='relu', padding='same',kernel_initializer=
'random_normal'),
  keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),
  keras.layers.Dropout(0.3),
  keras.layers.Conv2D(16, kernel_size=(3, 3),activation='relu', padding='same',kernel_initializer=
'random_normal'),
  keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),
  keras.layers.Conv2D(4 , (3, 3), activation='relu',padding='same',kernel_initializer='random_norm
al'),
  keras.layers.MaxPooling2D(pool_size=(2, 2),padding='same'),

  keras.layers.Conv2D(4 , (3, 3), activation='relu',padding='same',kernel_initializer='random_norm
al'),
  keras.layers.UpSampling2D((2, 2)),
  keras.layers.Conv2D(16, kernel_size=(3, 3),activation='relu', padding='same',kernel_initializer=
'random_normal'),
  keras.layers.UpSampling2D((2, 2)),
  keras.layers.Conv2D(32 , (3, 3), activation='relu',padding='same',kernel_initializer='random_nor
mal'),
  keras.layers.UpSampling2D((2, 2)),
  #keras.layers.Dropout(0.3),

  keras.layers.Conv2D(1 , (3, 3), activation='relu',padding='same',kernel_initializer='random_norm
al'),

])
print(autoencoderBASE .summary())
```

```
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_131 (Conv2D)          (None, 64, 312, 32)       320       
_________________________________________________________________
max_pooling2d_55 (MaxPooling (None, 32, 156, 32)       0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 32, 156, 32)       0         
_________________________________________________________________
conv2d_132 (Conv2D)          (None, 32, 156, 16)       4624      
_________________________________________________________________
max_pooling2d_56 (MaxPooling (None, 16, 78, 16)        0         
_________________________________________________________________
conv2d_133 (Conv2D)          (None, 16, 78, 4)         580       
_________________________________________________________________
max_pooling2d_57 (MaxPooling (None, 8, 39, 4)          0         
_________________________________________________________________
conv2d_134 (Conv2D)          (None, 8, 39, 4)          148       
_________________________________________________________________
up_sampling2d_53 (UpSampling (None, 16, 78, 4)         0         
_________________________________________________________________
conv2d_135 (Conv2D)          (None, 16, 78, 16)        592       
_________________________________________________________________
up_sampling2d_54 (UpSampling (None, 32, 156, 16)       0         
_________________________________________________________________
conv2d_136 (Conv2D)          (None, 32, 156, 32)       4640      
_________________________________________________________________
up_sampling2d_55 (UpSampling (None, 64, 312, 32)       0         
_________________________________________________________________
conv2d_137 (Conv2D)          (None, 64, 312, 1)        289       
=================================================================
Total params: 11,193
Trainable params: 11,193
Non-trainable params: 0
_________________________________________________________________
None
```

In [0]:

```python
# Load data
x_train = np.load(datapath +  'training_data.npy')
anomaly_data = np.load(datapath + 'test_data.npy')

x_train = x_train[:,0,:,:-1]
anomaly_data = anomaly_data[:,0,:,:-1]
```

In [0]:

```python
# Normal
#x_train = ( x_train - np.amin(x_train) )/( np.amax(x_train) - np.amin(x_train) )
#anomaly_data = ( anomaly_data - np.amin(anomaly_data) )/( np.amax(anomaly_data) - np.amin(anomaly
_data) )
test = anomaly_data

# reshape
normaldata_reshape = x_train.reshape(x_train.shape[0],64,312,1)
test_reshape = test.reshape(test.shape[0],64,312,1)

x_train_reshape = normaldata_reshape[:800,:,:,:]
x_val_reshape = normaldata_reshape[800:,:,:,:]
```

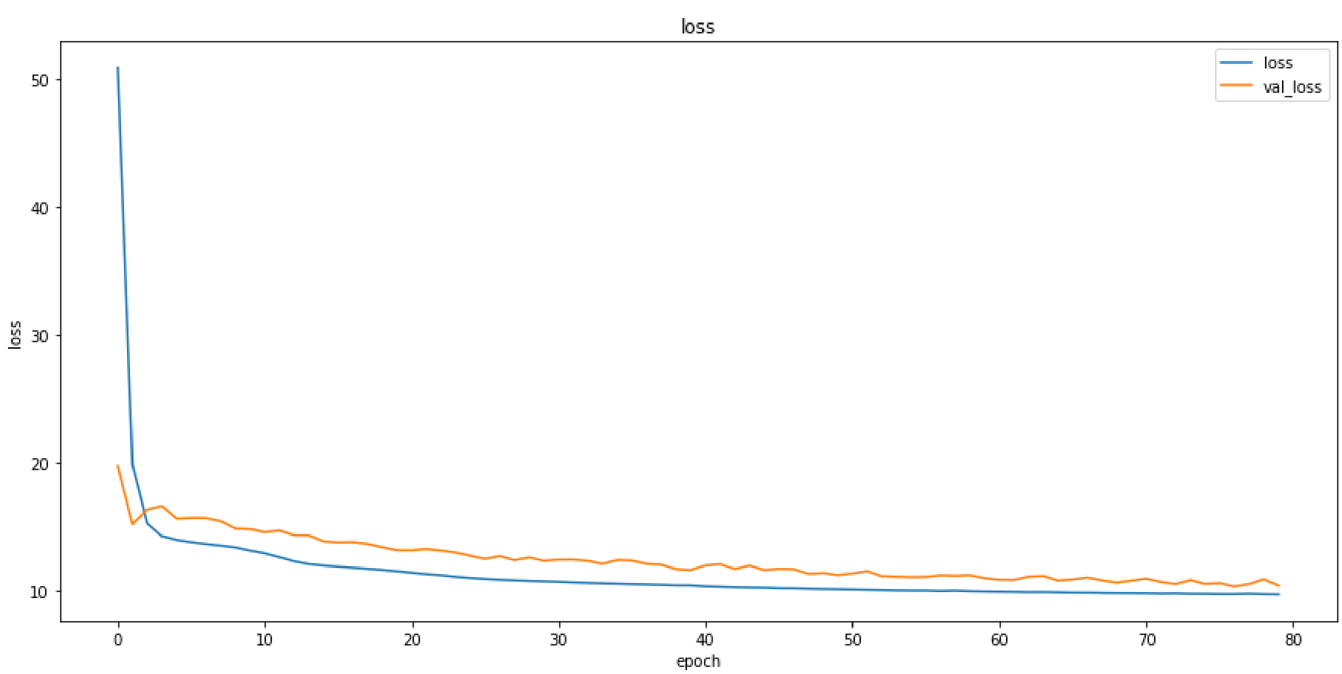
In [158]:

```
# Training
autoencoderBASE.compile(loss='mse', optimizer='adam')
history_conv_BASE = autoencoderBASE.fit(x_train_reshape, x_train_reshape, validation_data=(x_val_reshape, x_val_reshape), epochs=10, batch_size=50, shuffle=True)
decoded_data = autoencoderBASE.predict(test_reshape)
```

```
Epoch 1/10
16/16 [==============================] - 1s 64ms/step - loss: 12.1430 - val_loss: 11.2716
Epoch 2/10
16/16 [==============================] - 1s 57ms/step - loss: 10.0792 - val_loss: 10.4480
Epoch 3/10
16/16 [==============================] - 1s 58ms/step - loss: 9.7582 - val_loss: 10.8751
Epoch 4/10
16/16 [==============================] - 1s 58ms/step - loss: 9.6825 - val_loss: 10.7301
Epoch 5/10
16/16 [==============================] - 1s 58ms/step - loss: 9.6608 - val_loss: 10.6104
Epoch 6/10
16/16 [==============================] - 1s 58ms/step - loss: 9.6490 - val_loss: 10.5816
Epoch 7/10
16/16 [==============================] - 1s 57ms/step - loss: 9.6391 - val_loss: 10.5560
Epoch 8/10
16/16 [==============================] - 1s 58ms/step - loss: 9.6334 - val_loss: 10.5583
Epoch 9/10
16/16 [==============================] - 1s 58ms/step - loss: 9.6299 - val_loss: 10.5252
Epoch 10/10
16/16 [==============================] - 1s 58ms/step - loss: 9.6237 - val_loss: 10.5395
```

In [144]:

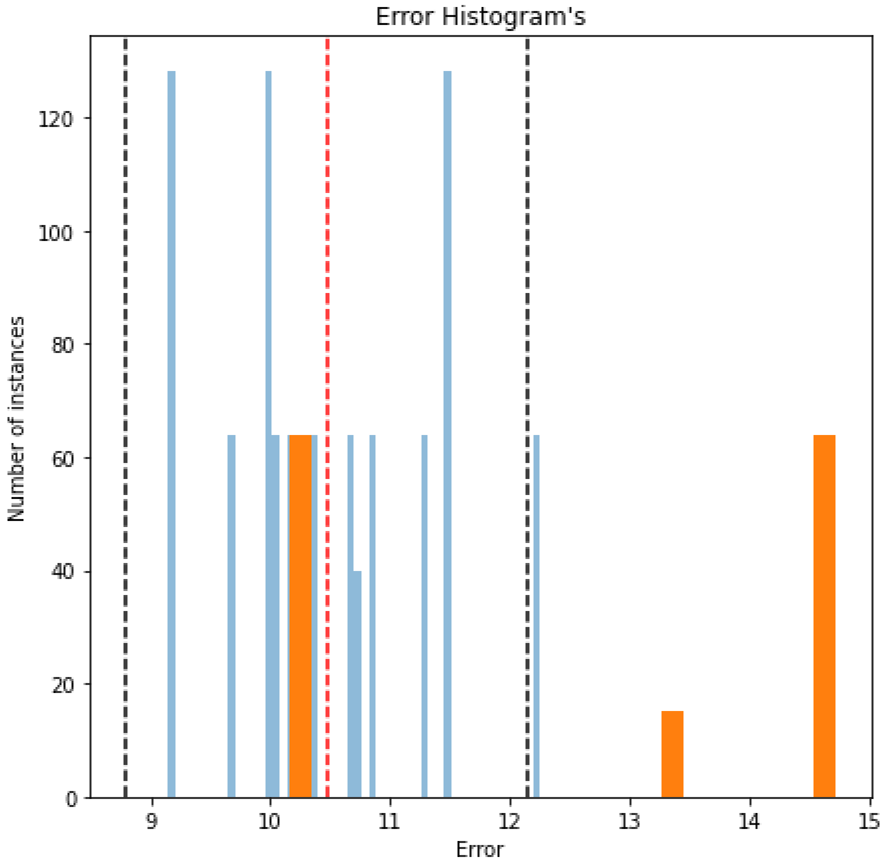
```
# Plot loss versus epoch.
plt.figure(figsize=(15,7))
plt.plot(history_conv_BASE.history['loss'],label='loss')
plt.plot(history_conv_BASE.history['val_loss'],label='val_loss')
plt.legend(loc='upper right')
plt.title('loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```



In [138]:

```
########## This code should remain untouched as much as possible,
#### except where your variable names for loss function or data set are needed.
#### This code feeds your data through the trained network to get mean and std
#### If you did not use a validation set then only use
#### your training data. Concatenating is therefore un-needed.
#### lossFunction <- Your loss function's name or use this one. Your choice of loss function.
##############################################################################################

lossFunction = tf.keras.losses.MeanSquaredError()

norm_list = []
dataset = (tf.data.Dataset.from_tensor_slices(normaldata_reshape)).batch(1)
for i, instance in dataset.enumerate():
    ae_predictions = autoencoderBASE(instance).numpy()
    norm_list.append(lossFunction(instance, ae_predictions).numpy())
# Feed the anomaly data through to get its error
anom_list = []
anomset = (tf.data.Dataset.from_tensor_slices(test_reshape)).batch(1)
for i, instance in anomset.enumerate():
    ae_predictions = autoencoderBASE(instance).numpy()
    anom_list.append(lossFunction(instance, ae_predictions).numpy())

normal_data_ERRORs = np.array(norm_list)
abnormal_data_ERRORs = np.array(anom_list)

######################### Code for presenting true positives to Question 2.3 ##################
threshold = 2
mean = normal_data_ERRORs.mean()
std = normal_data_ERRORs.std()
print(f'The mean of normal data is {mean:.4f}\
        and standard deviation is {std:.4f}')
upperbound = mean+threshold*std
lowerbound = mean-threshold*std
plt.figure(figsize=(7,7))
plt.title('Error Histogram\'s')
plt.hist(normal_data_ERRORs, bins=50, alpha=0.5)
plt.hist(abnormal_data_ERRORs, bins=25, alpha=1.0)
plt.axvline(mean,ls='--', c='r')
plt.axvline(lowerbound, ls='--',c='k')
plt.axvline(upperbound, ls='--',c='k')
plt.xlabel('Error')
plt.ylabel('Number of instances')
plt.show()
```

```
The mean of normal data is 10.4711        and standard deviation is 0.8402
```



In [139]:
```python
tp_count = np.sum(abnormal_data_ERRORs >= upperbound) +\
           np.sum(abnormal_data_ERRORs <= lowerbound)
fn_count = anomaly_data.shape[0] - tp_count
fp_count = np.sum(normal_data_ERRORs >= upperbound)+\
           np.sum(normal_data_ERRORs <= lowerbound)
tn_count = 1000 - fp_count

print(f'TP {tp_count} \t FP {fp_count}')
print(f'FN {fn_count} \t TN {tn_count}')
```

```
TP 79    FP 64
FN 64    TN 936
```