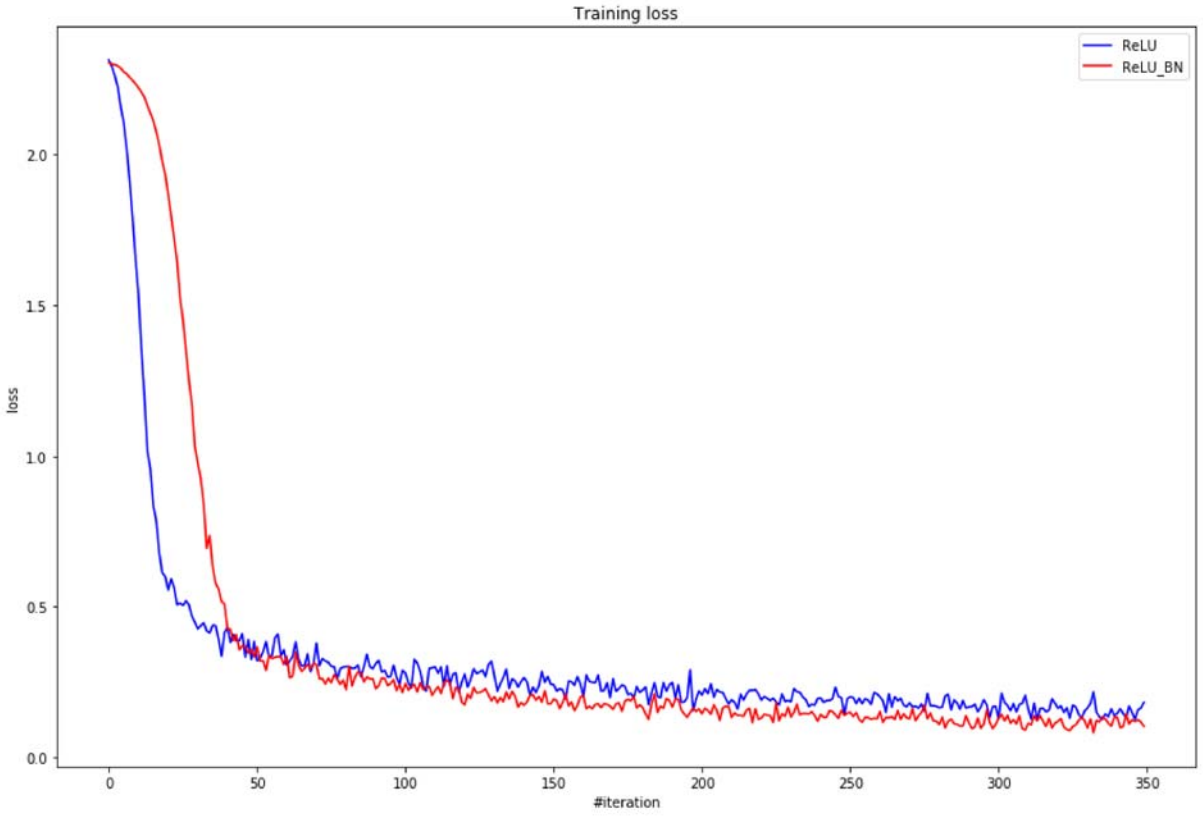
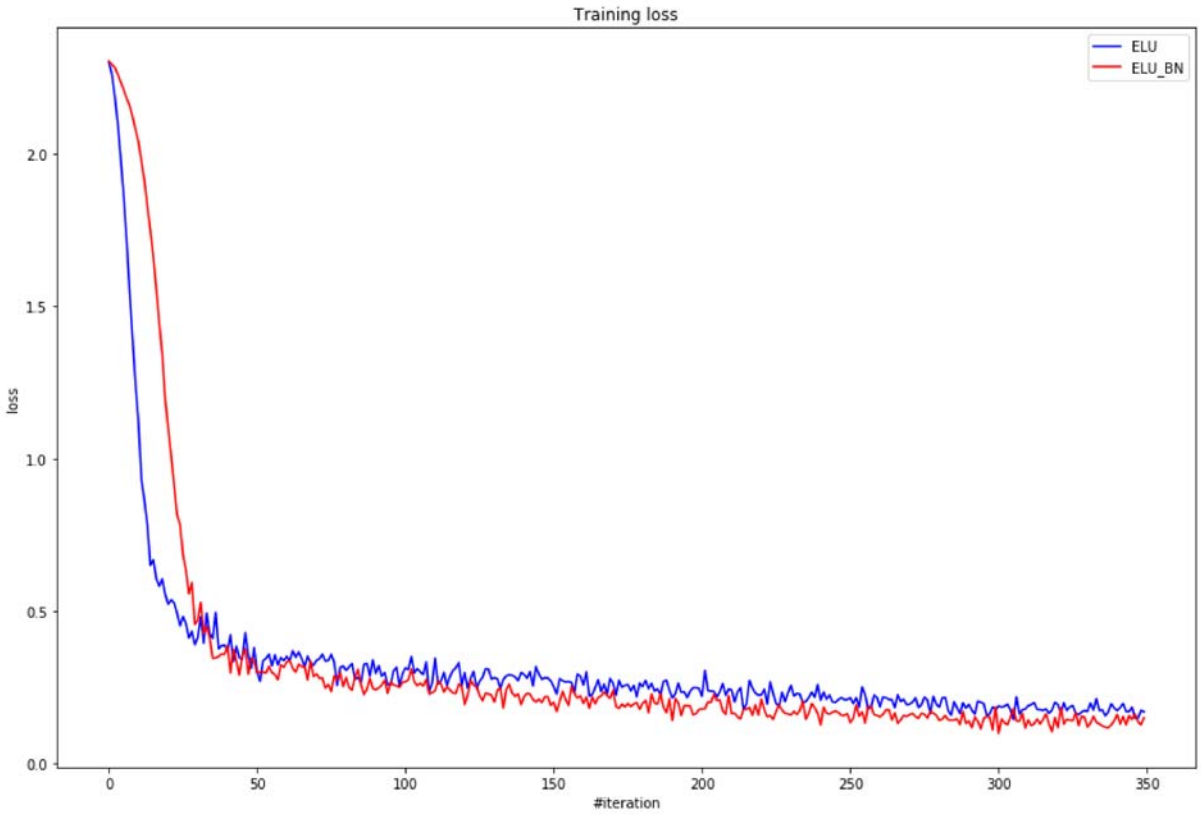
In [64]:
```
print("Number of neurons in the inner layer: ", node_size)
for (a, l, bn_l) in losses:
    # Visualize
    plt.figure(figsize=(15, 10))

    plt.title("Training loss")
    plt.xlabel("#iteration")
    plt.ylabel("loss")
    line = plt.plot(l, 'b', label=a)
    line_bn = plt.plot(bn_l, 'r', label=a + "_BN")

    plt.legend(loc="best")
    plt.show()

    print(a," loss: ", np.min(l))
    print(a," loss BatchNorm : ", np.min(bn_l))
```

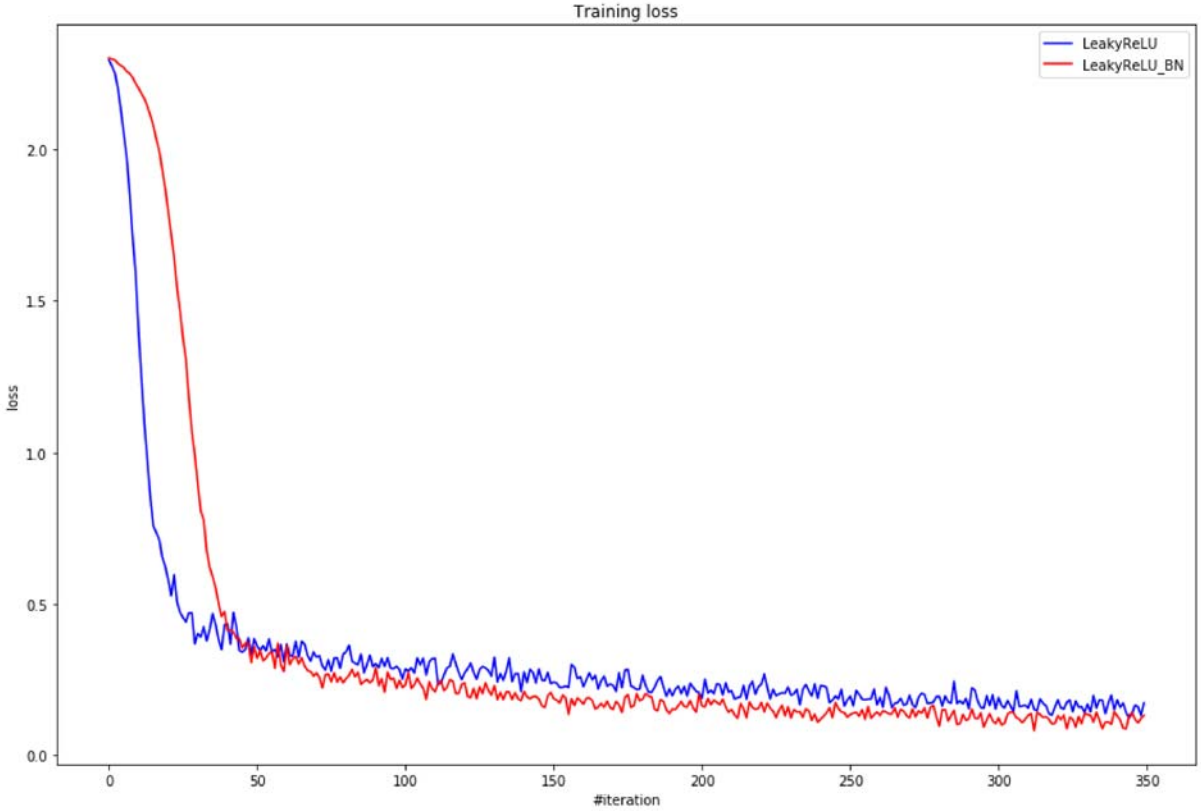
```
Number of neurons in the inner layer:  100
```



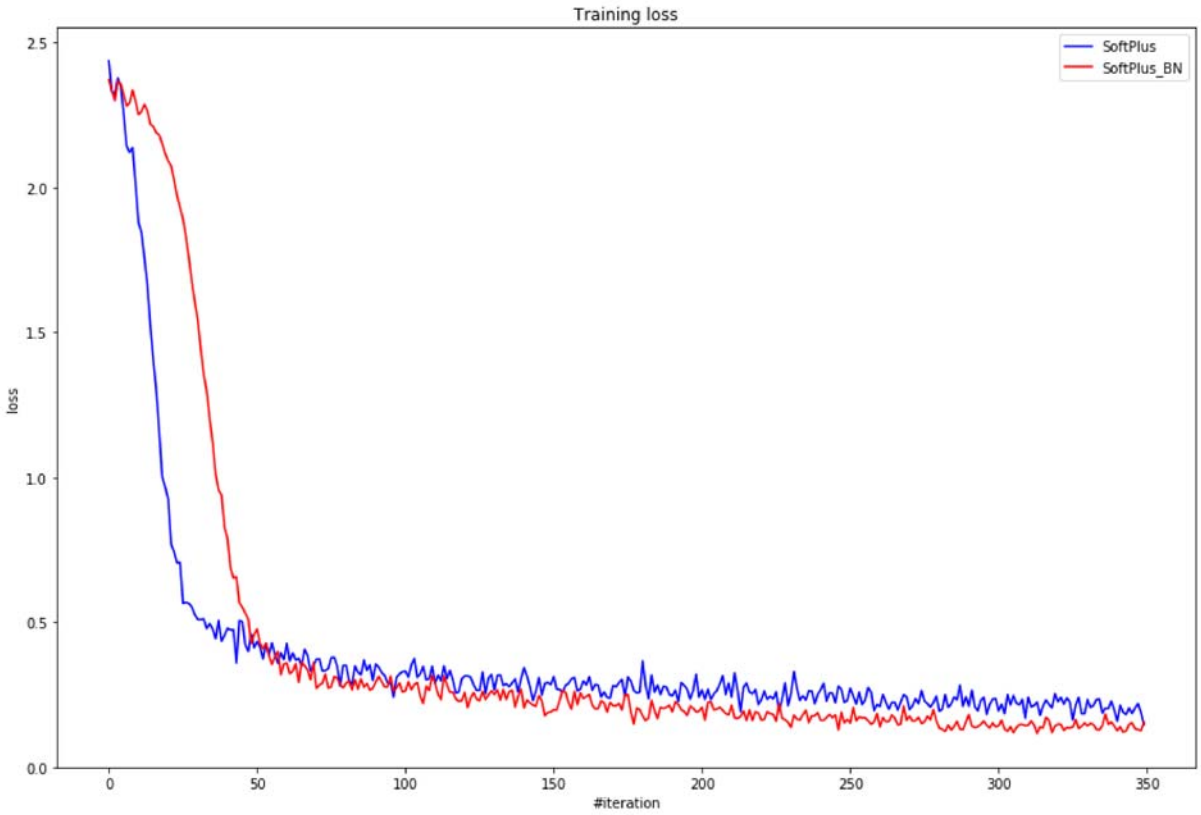

```
ReLU  loss:  0.1240531751966891
ReLU  loss BatchNorm :  0.08146345569564155
```




```
ELU  loss:  0.14530047628116777
ELU  loss BatchNorm :  0.09927520319417292
```




```
LeakyReLU  loss:  0.12140542995232508
LeakyReLU  loss BatchNorm :  0.08023169995481644
```




```
SoftPlus  loss:  0.14539744725439027
SoftPlus  loss BatchNorm :  0.11501018352457609
```

In [ ]:

Write your personal opinion on the activation functions, think about computation times too. Does `BatchNormalization` help?

In [13]:
```
# Your answer goes here. ################################################
```

Finally, use all your knowledge to build a super cool model on this dataset, do not forget to split dataset into train and validation. Use **dropout** to prevent overfitting, play with **learning rate decay**. You can use **data augmentation** such as rotations, translations to boost your score. Use your knowledge and imagination to train a model. Don't forget to call `training()` and `evaluate()` methods to set desired behaviour of `BatchNormalization` and `Dropout` layers.

In [14]:
```
# Your code goes here. ################################################
```

Print here your accuracy. It should be around 90%.

In [15]:
```
# Your answer goes here. ################################################
```

Autoencoder