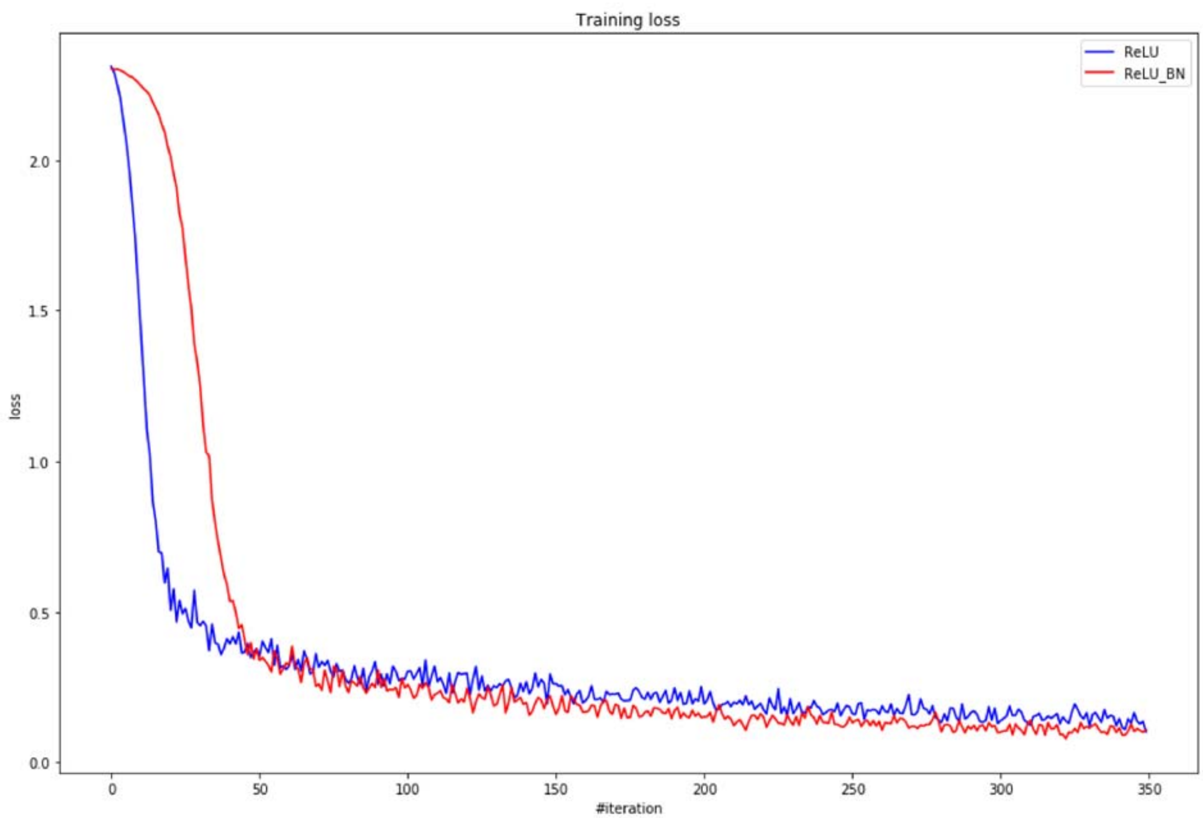
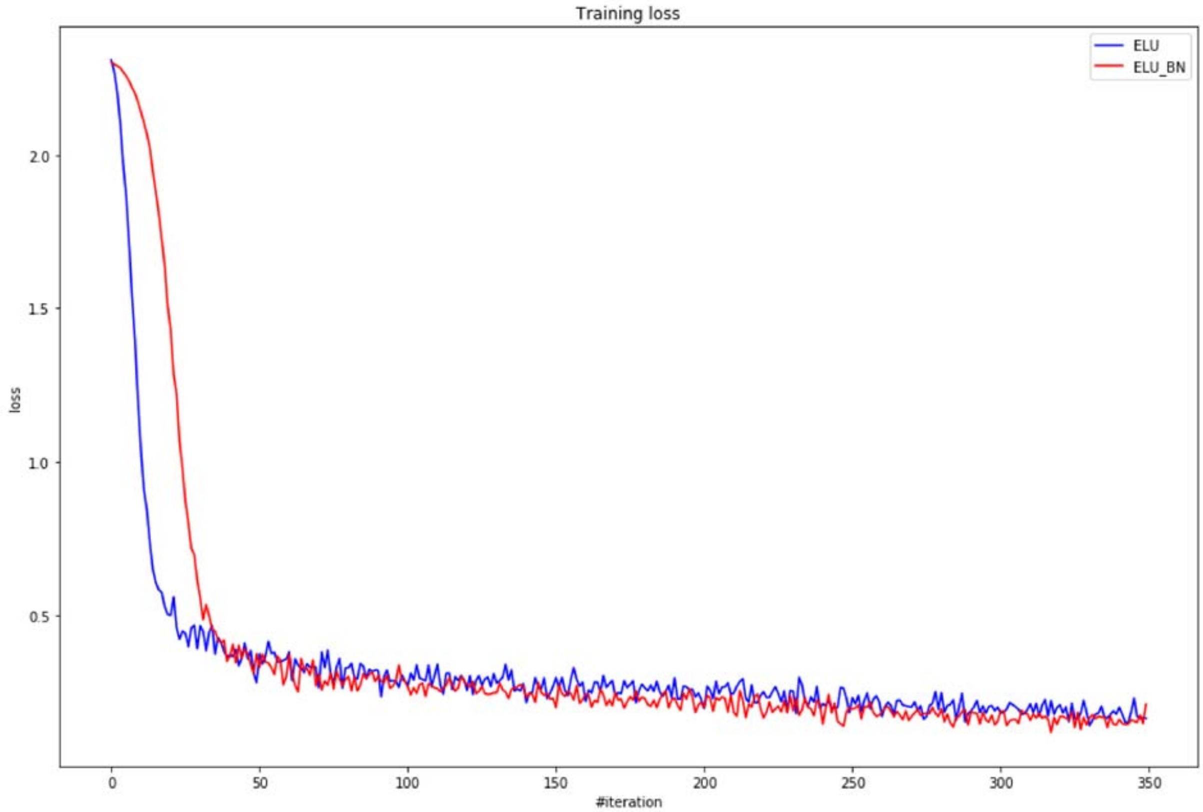
In [66]:

```
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))
```

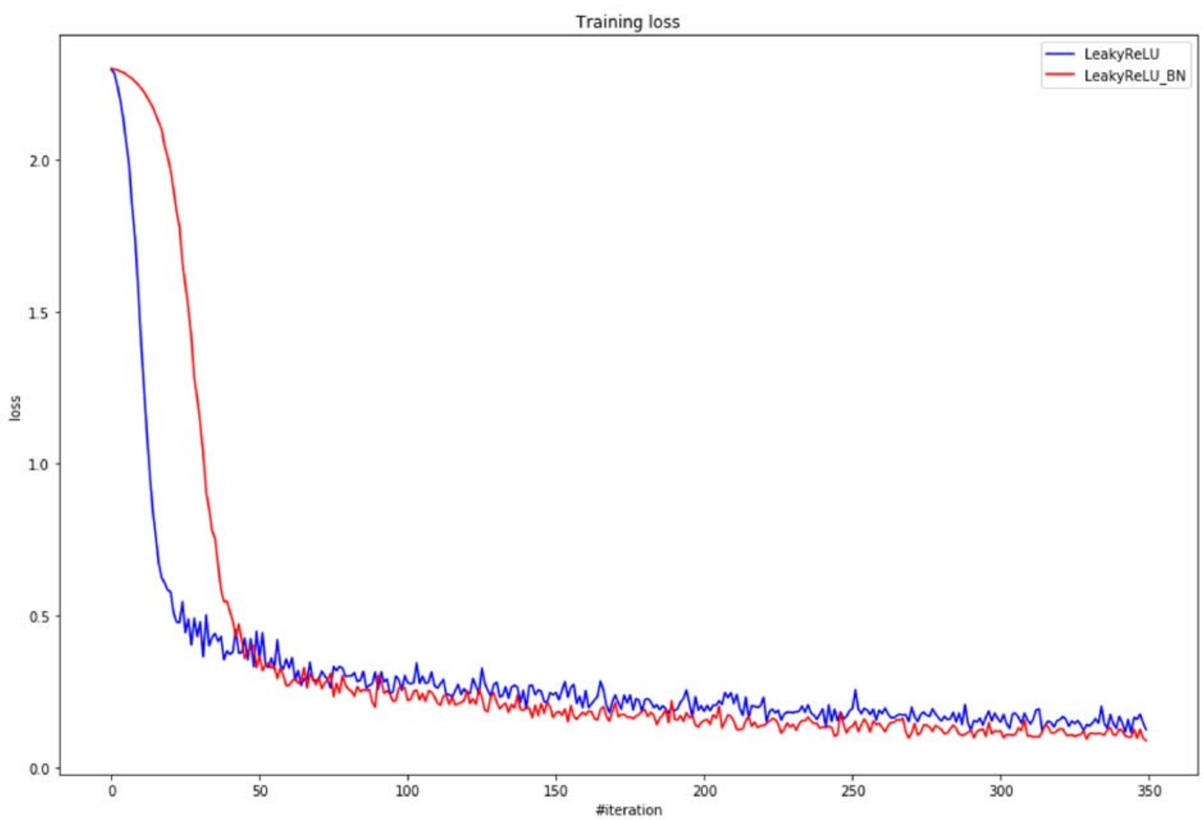
```
Number of neurons in the inner layer:  200
```



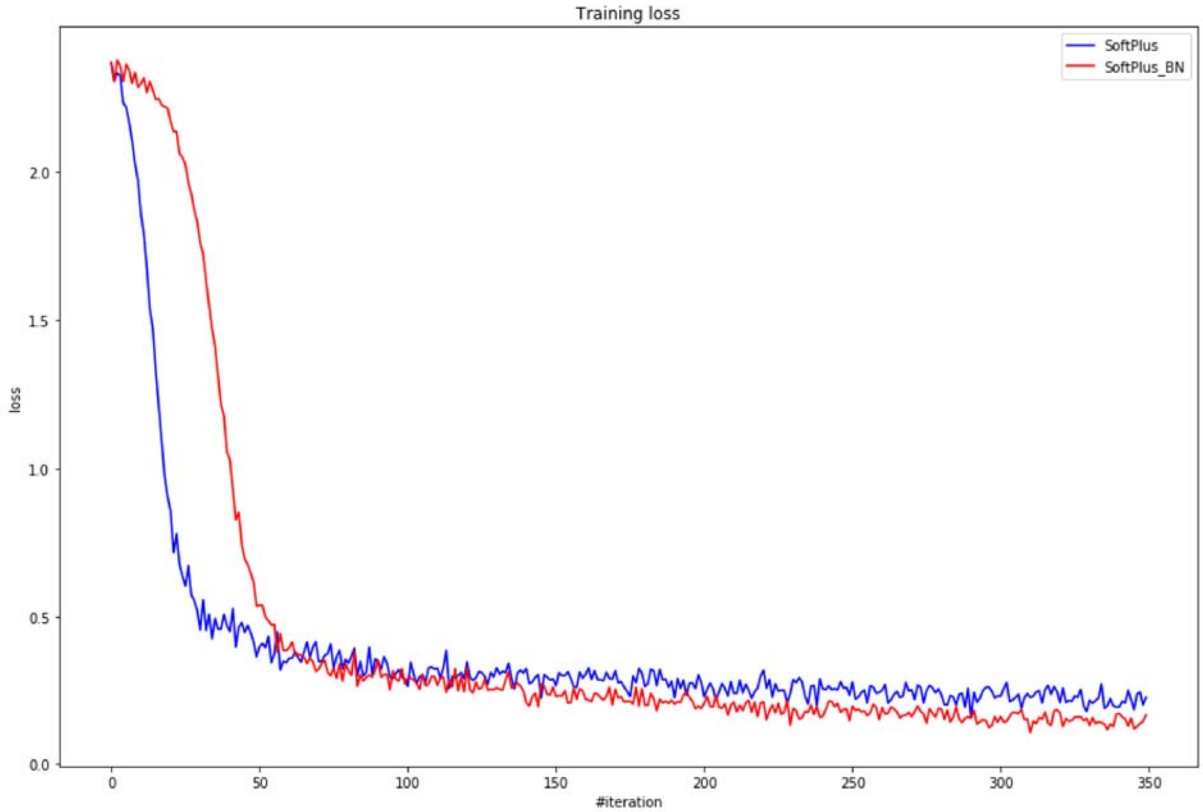

```
ReLU  loss:  0.10053982531142128
ReLU  loss BatchNorm :  0.07718396767398226
```




```
ELU  loss:  0.13873209248259433
ELU  loss BatchNorm :  0.11674414394265792
```




```
LeakyReLU  loss:  0.11341953401912494
LeakyReLU  loss BatchNorm :  0.08772101119969121
```




```
SoftPlus  loss:  0.1584598205432189
SoftPlus  loss BatchNorm :  0.10737357330850125
```

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

- BatchNorm yields lower loss and leads to a smoother descent (not faster, but more steady descent)
- LeakyReLU and ReLU are the best performing nonlinearities
- More nodes in the inner layer lead to smaller loss
- There exists a "smooth spot" where increasing the number of inner nodes does not give more performance. Therefore we can say that the performance is given by the available data, and not the number of neurons

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%.