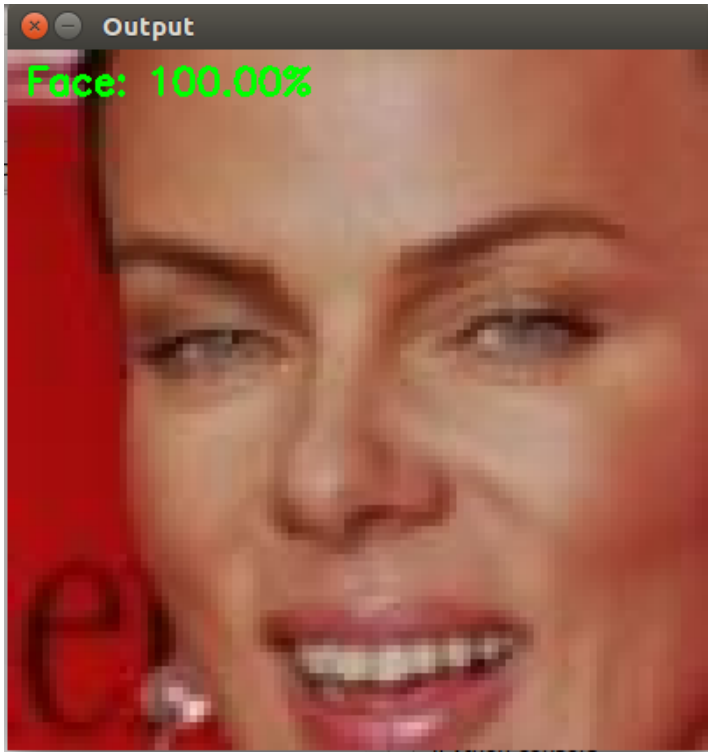
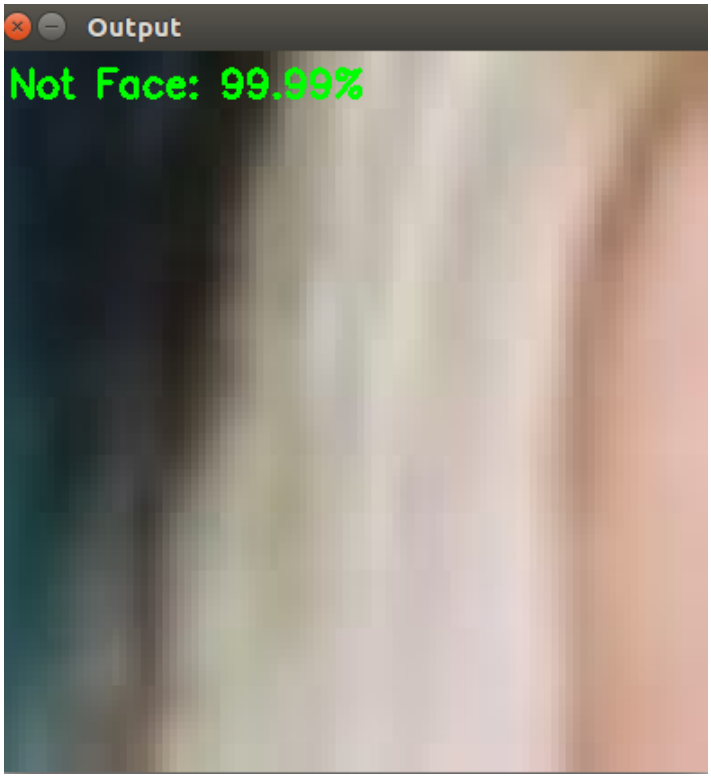
I have used a LeNet model designed using keras(backend Tensorflow) as my network architecture, with two convolution layers.

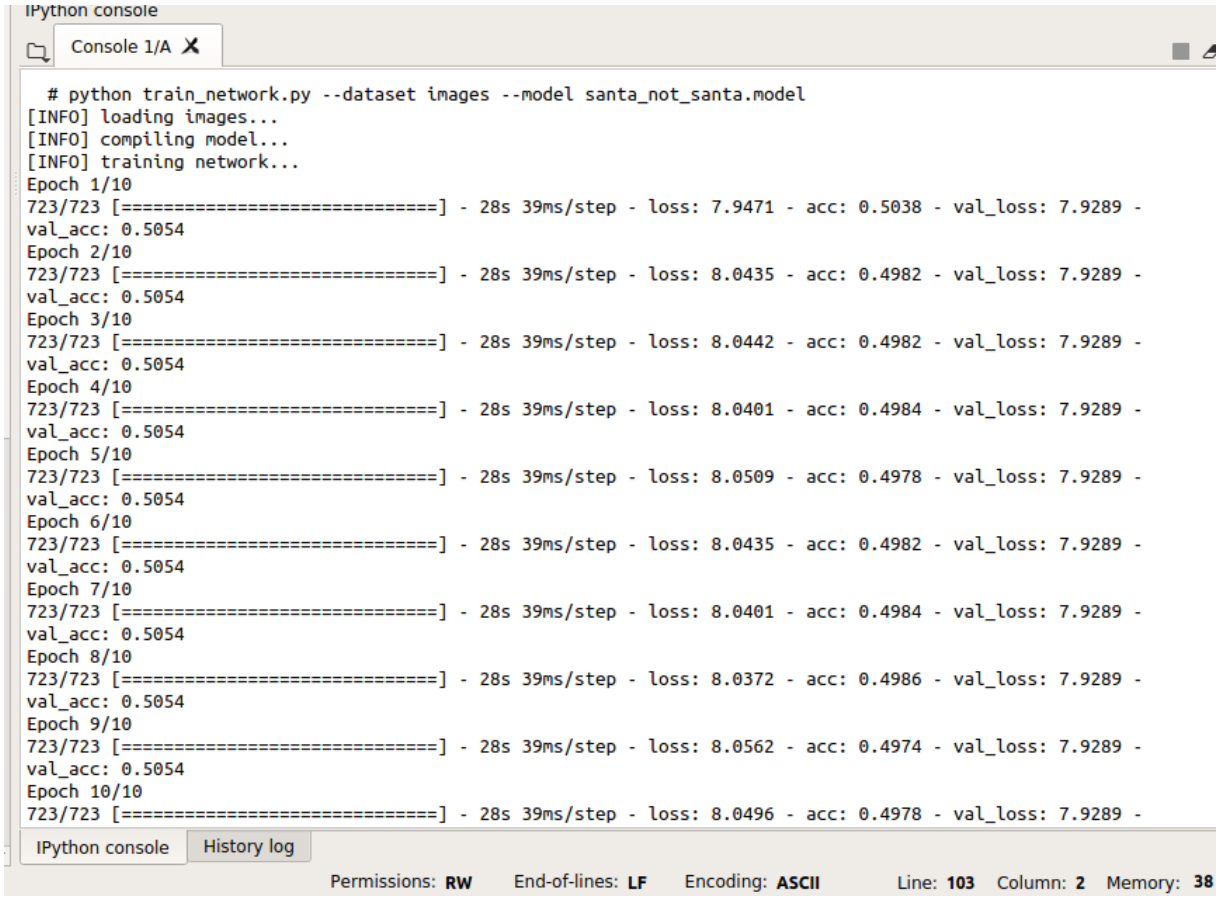
Some Examples



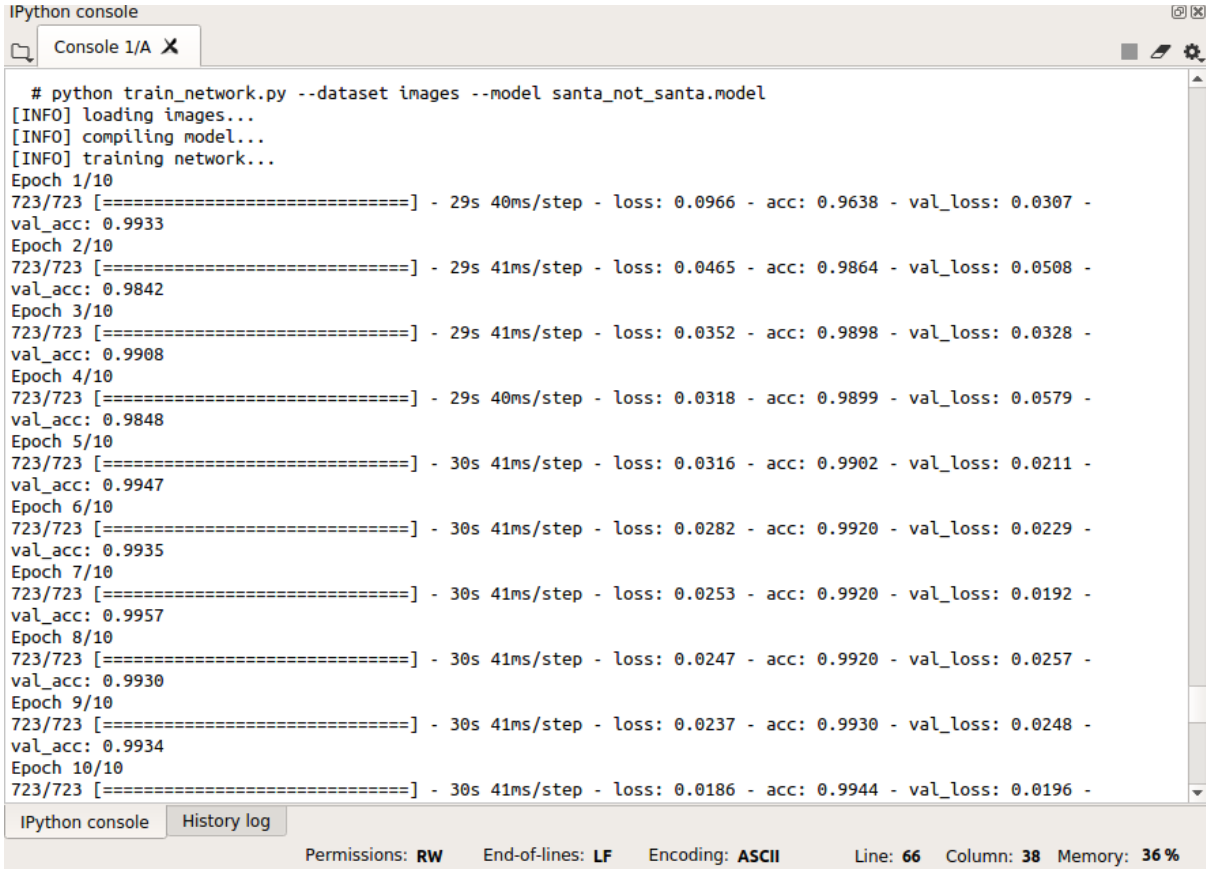


Preprocessing the data

Without Normalizing the data, the results are as shown below:



After Normalizing the data, the results are as shown below:



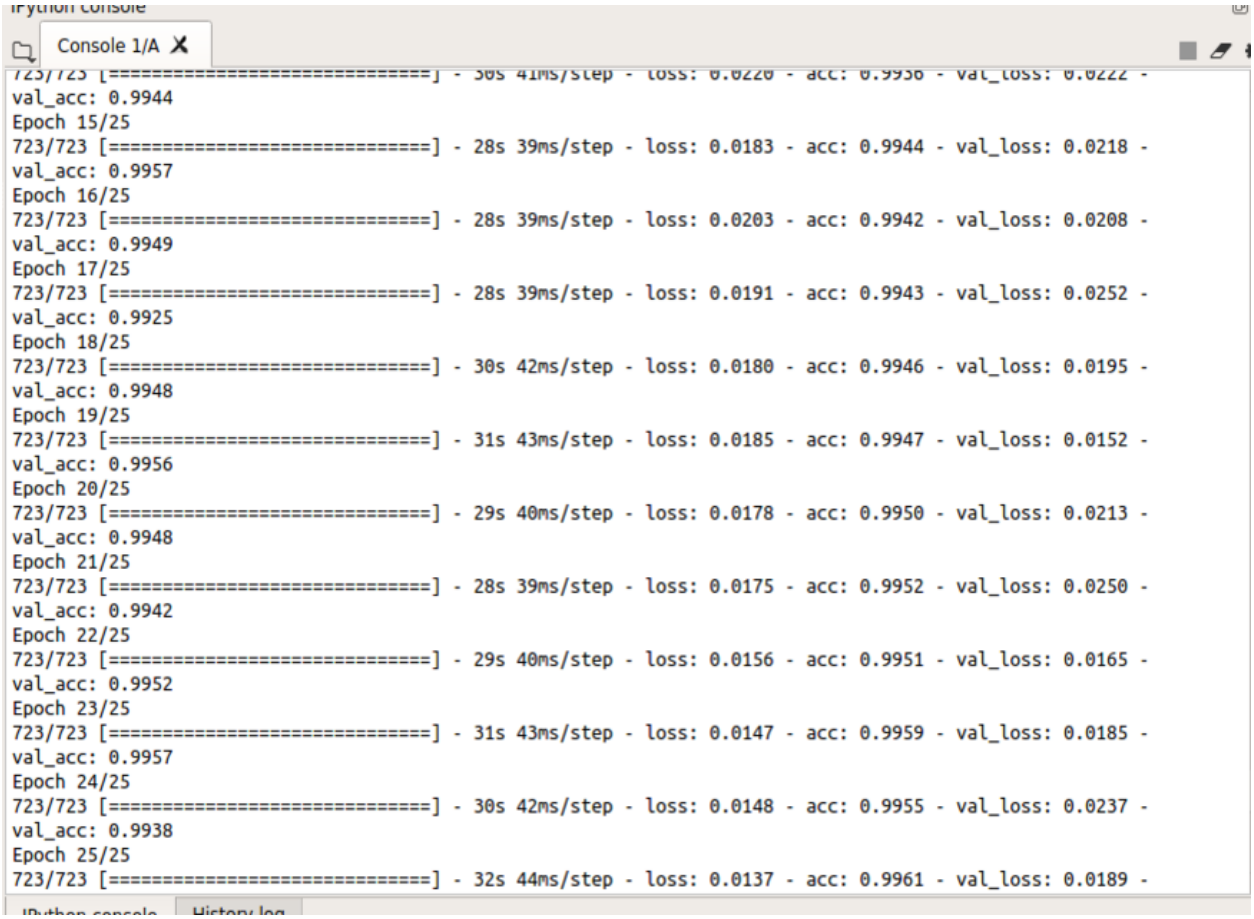
As we can clearly see, normalized helped us in achieving a lower loss and we obtained a higher validation accuracy.

BabySitting Process

Checked that the loss is reasonable, while disabling the regularization as shown below:

```
        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(500,kernel_regularizer=regularizers.l2(0.00)))
        model.add(Activation("relu"))
```

The loss obtained after training the model is pretty low, as expected. But this might have a tendency to overfit the model.



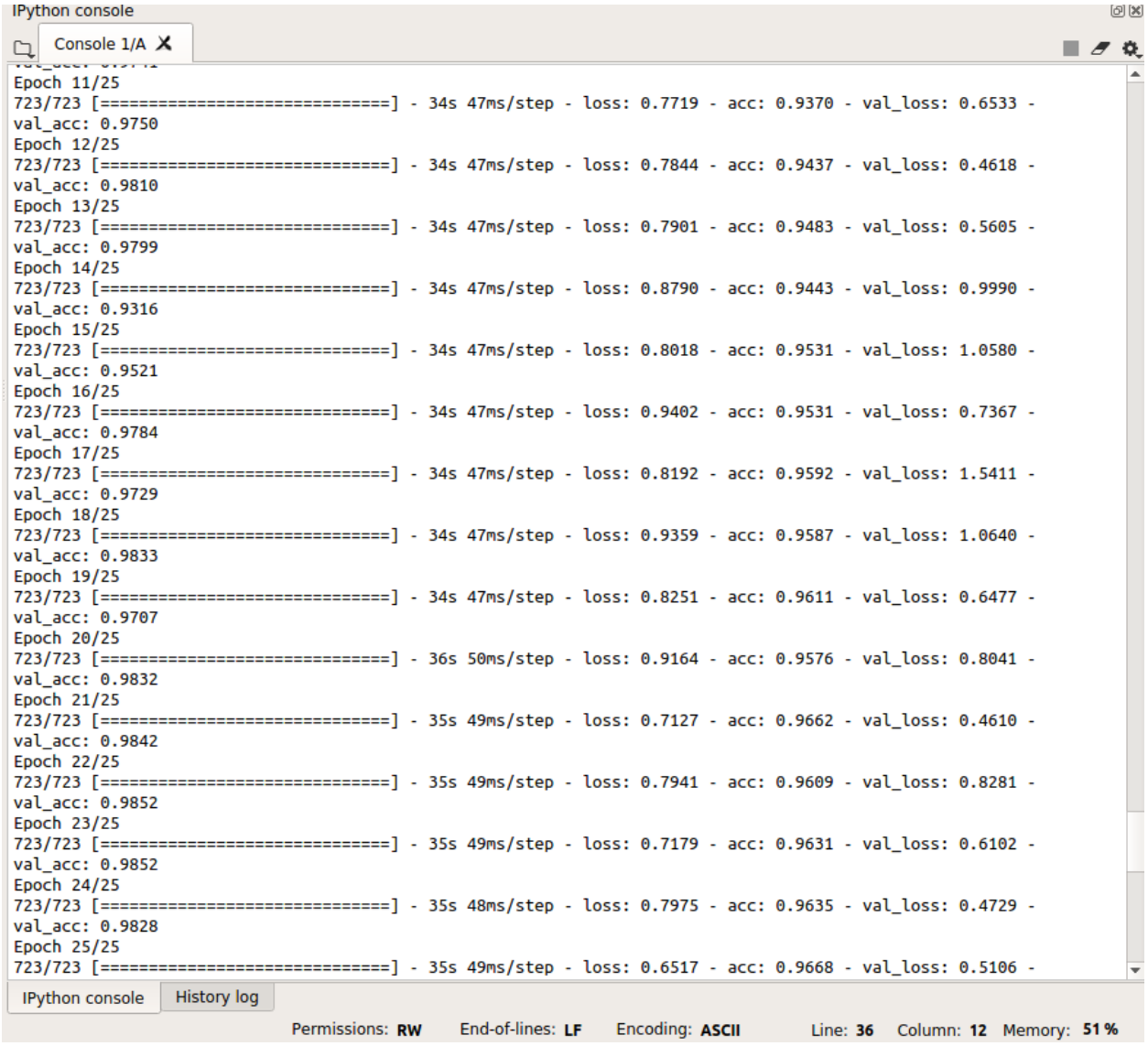
So, we crank up the regularization to 1e3:

```
# initialize the model
print("[INFO] compiling model...")
model = LeNet.build(width=28, height=28, depth=3, classes=2,regular = 1e3)
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# train the network
print("[INFO] training network...")
H = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=1)
```

And as expected, the loss goes up significantly, as expected.

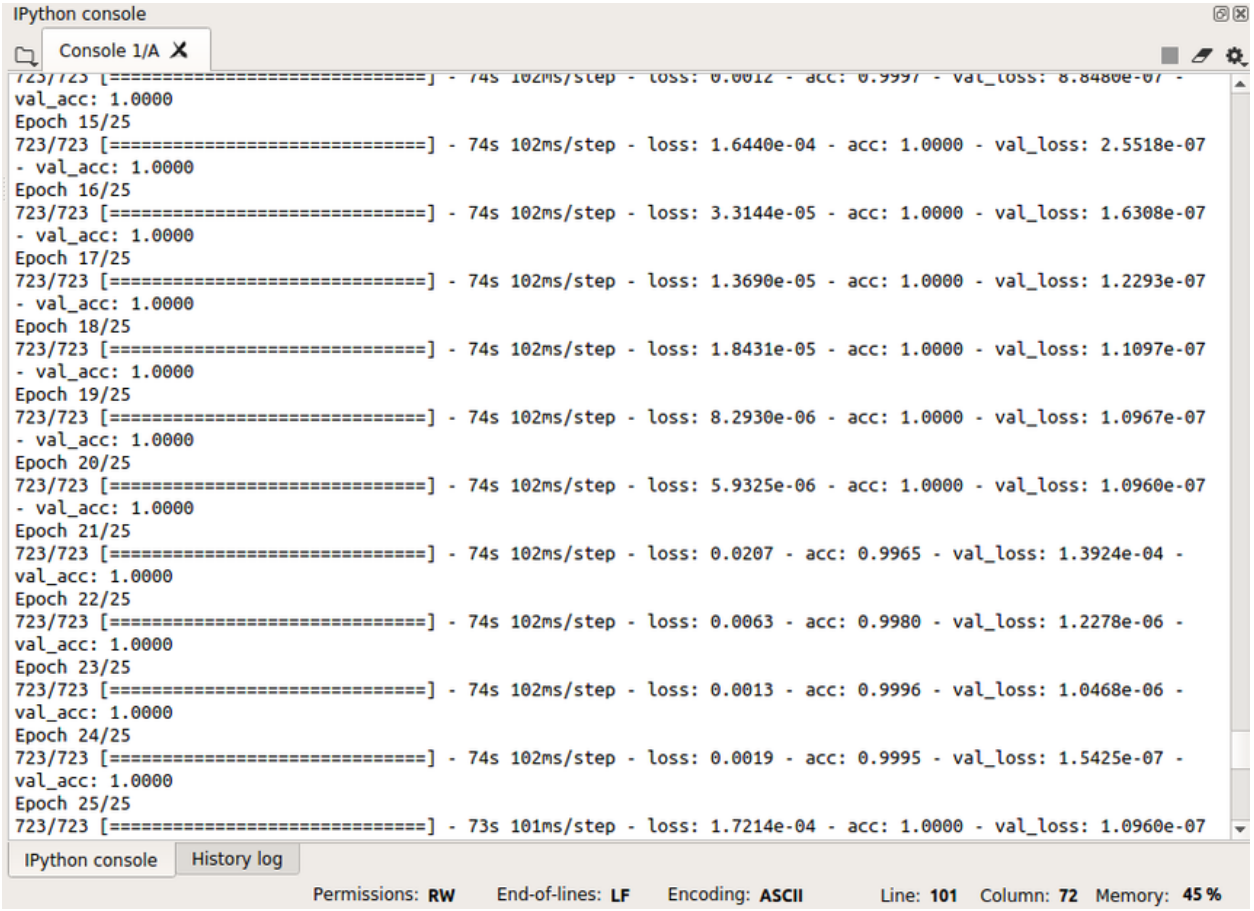
(25th Epoch - loss 0.0137 vs loss 0.6517)



Next, we try to overfit a very small portion of the training data. I took the first 20 examples of the the face non face training data.

```
best_model = model.fit_generator(aug.flow(tiny_X, tiny_Y, batch_size=BS),
    validation_data=(tiny_X, tiny_Y), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=1)
```

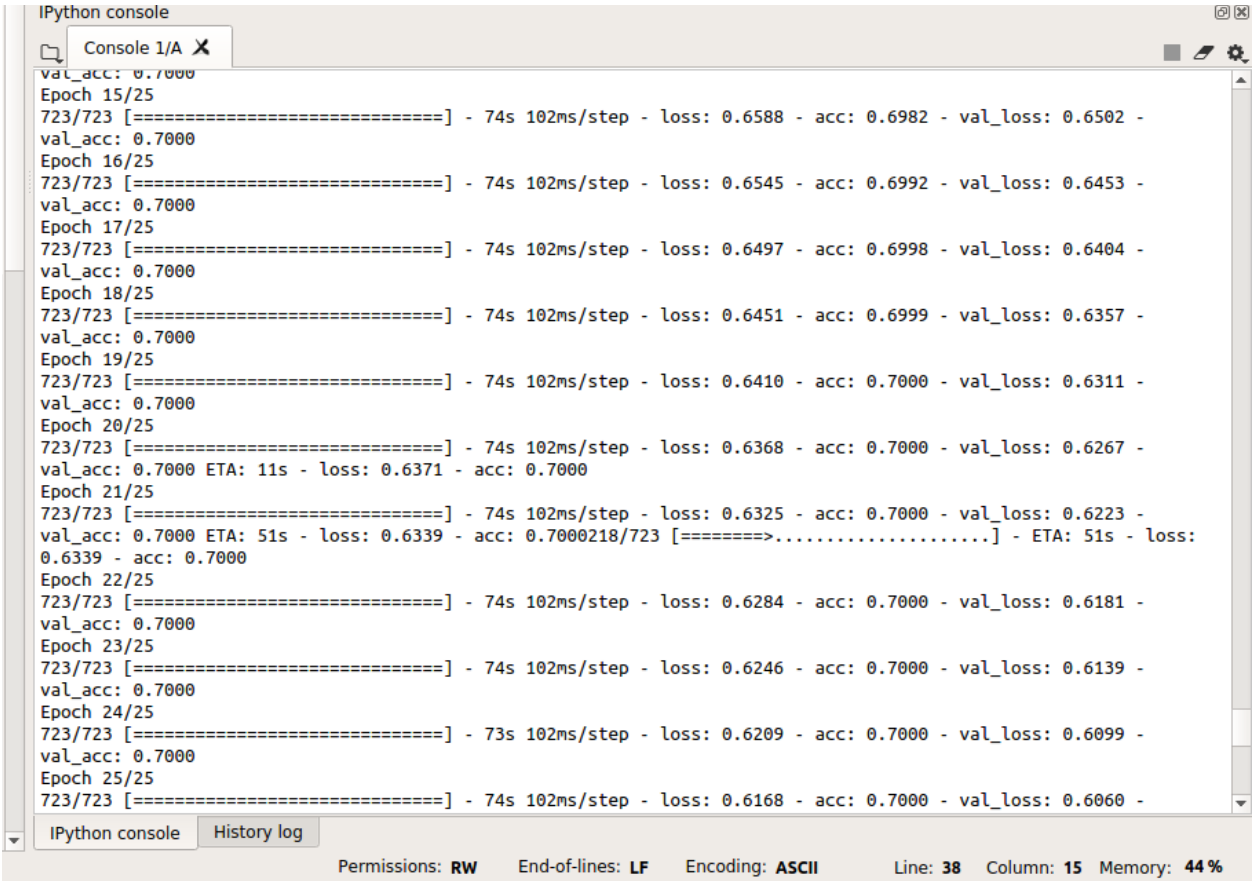
And as expected, we overfitted the model over the data(as we can see that the accuracy is exactly 1) and the loss per epoch is very close to 0(in the magnitude of 1e-6).



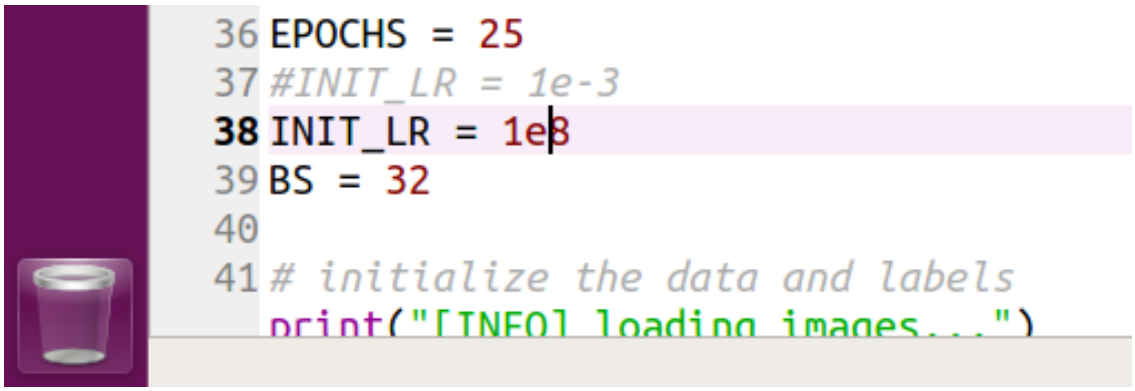
Now, we start with a small learning rate and regularization that makes the loss go down. I set the value to 1e-8 as shown below:

```
36 EPOCHS = 25
37 #INIT_LR = 1e-3
38 INIT_LR = 1e-8
39 BS = 32
40
```

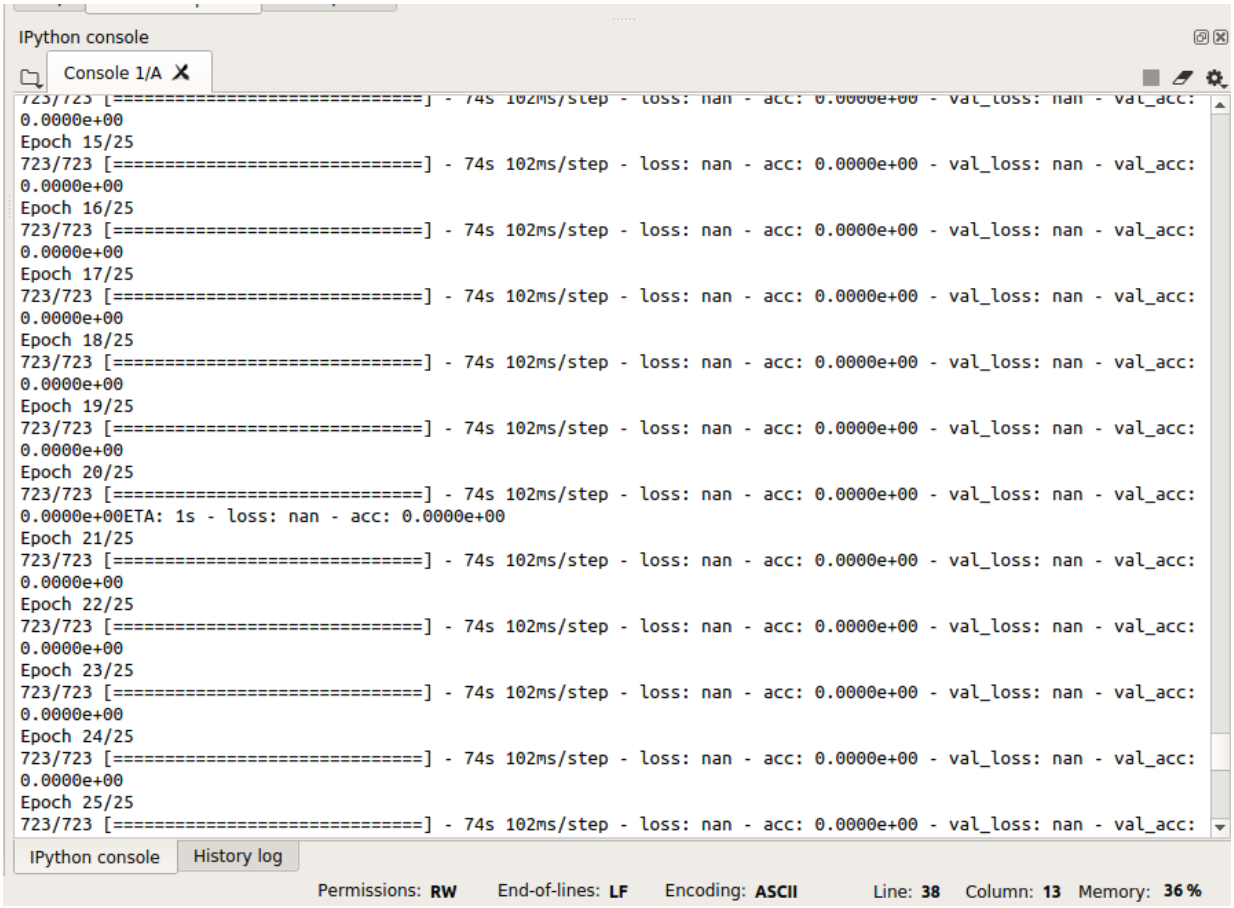
As we can see, the loss is barely changing from 0.6. The model improves very slowly.



We then try the other extreme by setting the learning rate to 1e8 as shown below:

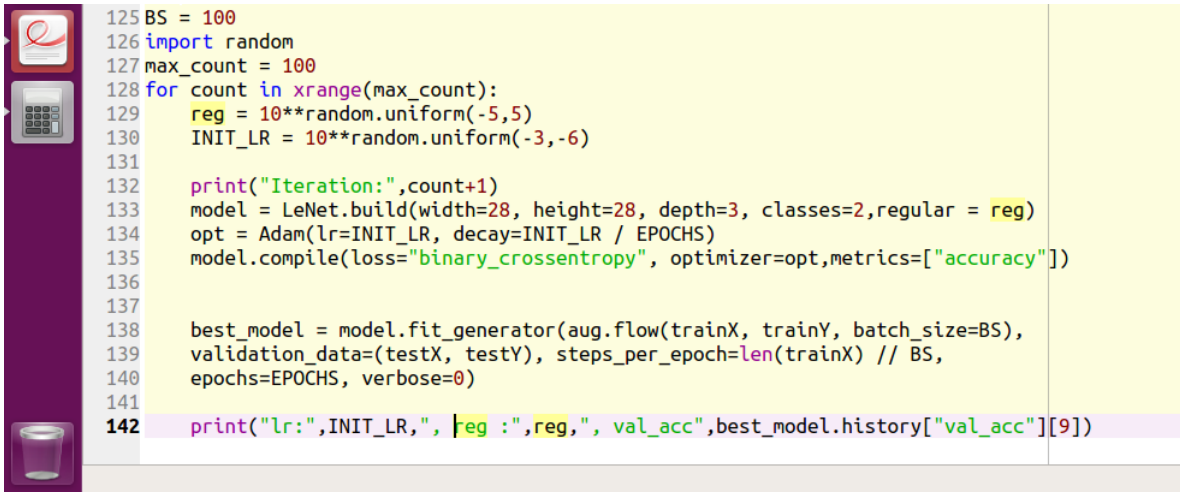


As we set the learning rate too high, we can see that the loss is exploding and we get a couple of NaNs as shown. This is an indication of high learning rate.



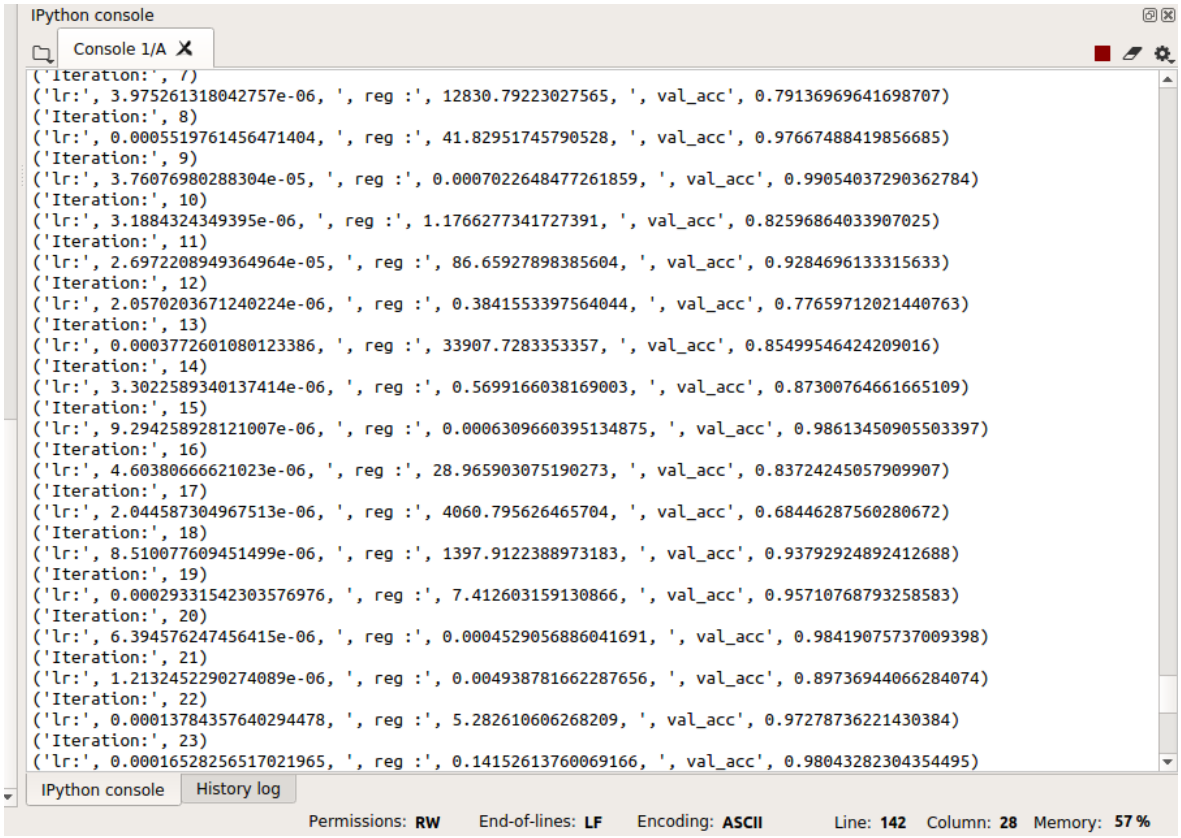
Hyperparameter Optimization

First we run a coarse search of 10 epochs each to get a rough estimate of the range in which the regularization and learning rate might be in. It is as shown below:



```
125 BS = 100
126 import random
127 max_count = 100
128 for count in xrange(max_count):
129     reg = 10**random.uniform(-5,5)
130     INIT_LR = 10**random.uniform(-3,-6)
131
132     print("Iteration:",count+1)
133     model = LeNet.build(width=28, height=28, depth=3, classes=2,regular = reg)
134     opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
135     model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])
136
137
138     best_model = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
139     validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
140     epochs=EPOCHS, verbose=0)
141
142     print("lr:",INIT_LR,", reg :",reg,", val_acc",best_model.history["val_acc"][9])
```

As we can clearly see, we get a good accuracy at Iteration 9 and 22, where the learning rate is around 1e-5 - 0.0005 and the regularization is close to 0 for a good accuracy



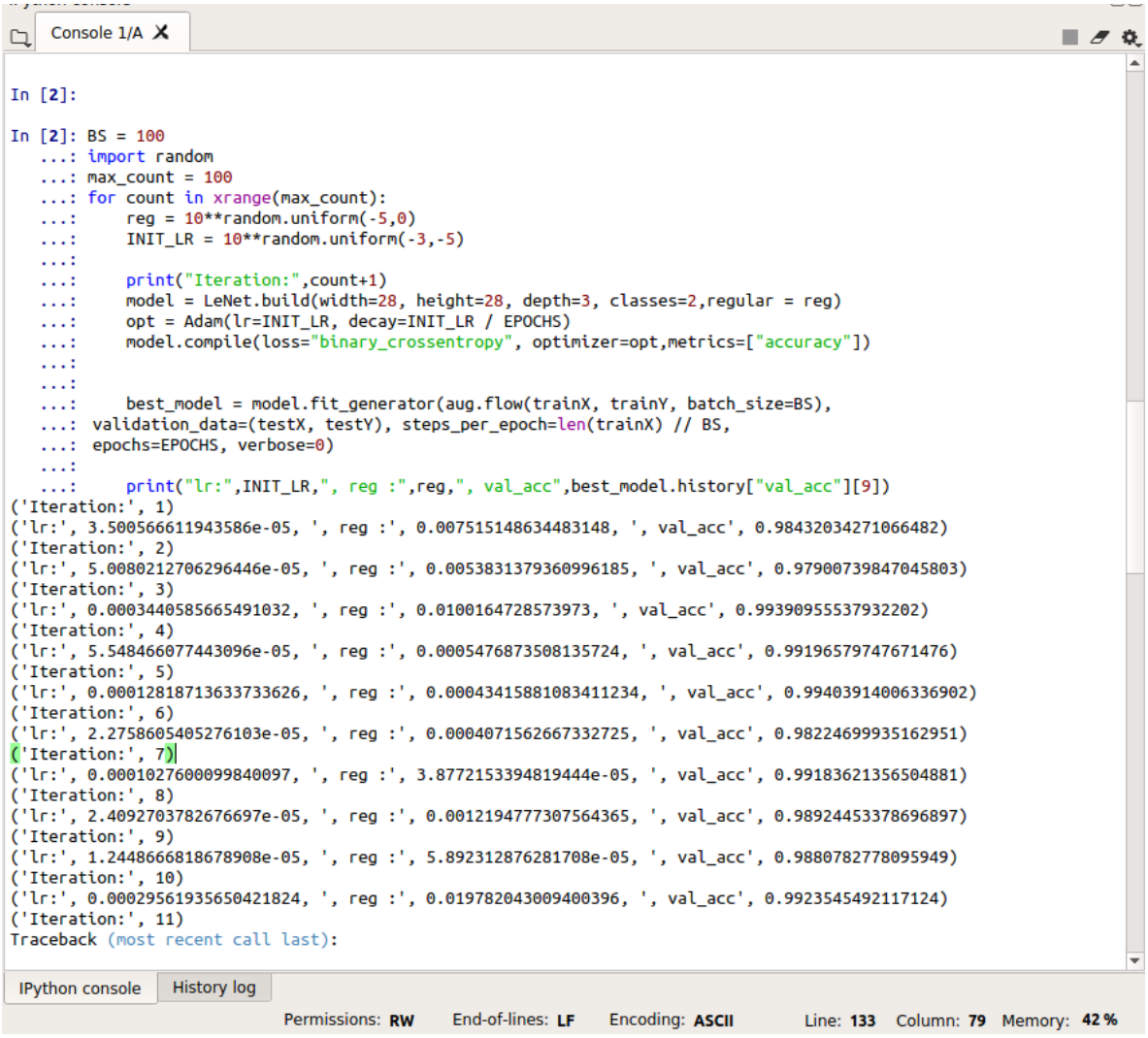
So we run a fine search by changing the random initialization of the learning rates and regularizations to a tighter range, as shown below:

```
#%%
BS = 100
import random
max_count = 100
for count in xrange(max_count):
    reg = 10**random.uniform(-5,0)
    INIT_LR = 10**random.uniform(-3,-5)

    print("Iteration:",count+1)
    model = LeNet.build(width=28, height=28, depth=3, classes=2,regular = reg)
    opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
    model.compile(loss="binary_crossentropy", optimizer=opt,metrics=["accuracy"])


    best_model = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
    validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
    epochs=EPOCHS, verbose=0)

    print("lr:",INIT_LR,", reg :",reg,", val_acc",best_model.history["val_acc"][9])
```



As we can see the best accuracy is obtained for:

Learning rate: 0.0001281871

Regularization:0.0004341588

Accuracy obtained : 99.4%