UPPSALA UNIVERSITET

COMPLEX DATA: ANALYSIS & VISUALISATION

Exercise 4

Deep Learning

EFTHYMIA CHANTZI

MSc in Bioinformatics

Email: Efthymia.Chantzi.0787@student.uu.se

January 5, 2016

MATLAB R2015b and the Neural Network Toolbox^{TM} [1] is used for the fulfillment of this assignment. The documented code that pertains to the implementation of the following tasks is accessible under Deep-Learning/Exercise4 at:
https://github.com/EffieChantzi/Deep-Learning.git.

1 Tasks A, D

These two tasks constitute a more successful implementation of the previous assignment, where *deep networks* are trained with the goal of image reconstruction. More precisely, the results gained by the previous assignment indicated that the hidden layer's structure $(25 - 3 - 25)$, was not adequate for a visually acceptable reconstruction with the same or at least approximately the same mean squared error, as obtained by applying plain PCA.

Consequently, in this exercise, the same approach is used but with increased number of neurons in the hidden layers. The employed initialization methods for weights and biases of the deep networks remain the same as before; *principal component analysis* and *ordinary stacked autoencoders*. Moreover, the initialization approach based on denoising stacked autoencoders (Task D) is implemented and tested against the other two. The training and test datasets consist of 5000 digit images with 784 pixels (digittrain_dataset, digittest_dataset).

Provided that three different implementations of deep learning must be compared, common training settings are used, which are stated in table (1):

Training Settings	
Algorithms	
Data Division	Random (*dividerand*)
Training Function	Scaled Conjugate Gradient (*trainscg*)
Performance Function	Mean Square Error (*mse*)
Termination Criteria	
Maximum Number of Training Iterations (*max_epochs*)	1500
Minimum Gradient Magnitude (*min_grad*)	10^{-6}
Maximum Number of Validation Increases (*max_fail*)	6

Table 1: *Common training settings and parameters used for all deep networks of this exercise*

1.1 PCA Initialization

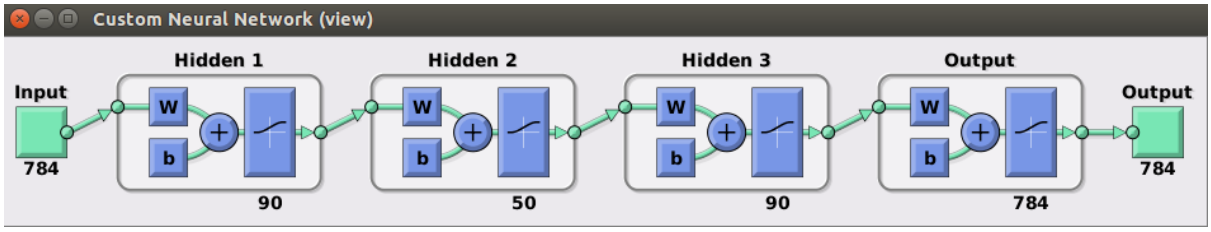


Figure 1: *Deep Neural Network 784-90-50-90-784 with PCA initialization*

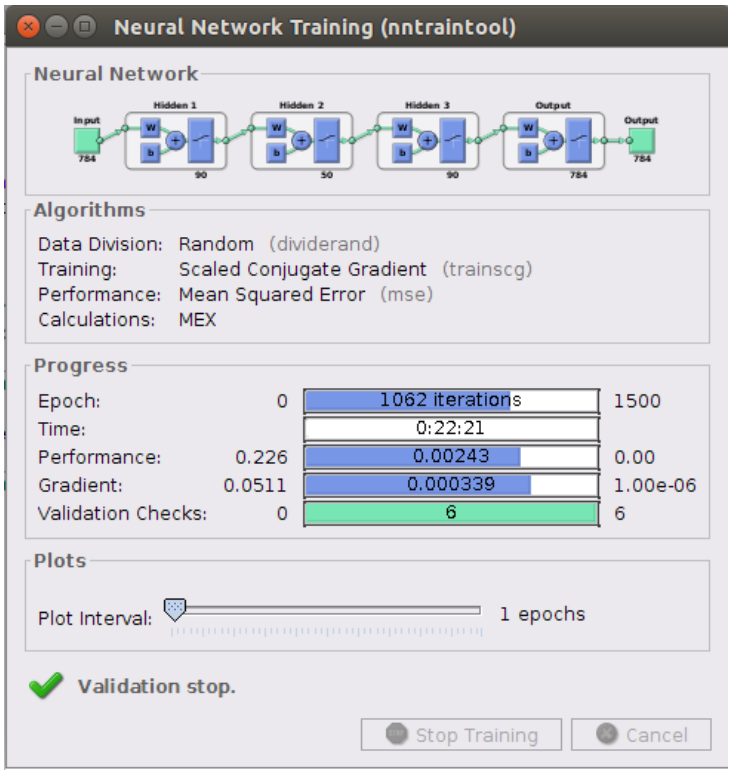


Figure 2: *Training window*

Time (hrs-min-sec)	
Training	00:22:21

Table 2: *Time requirements on local machine*

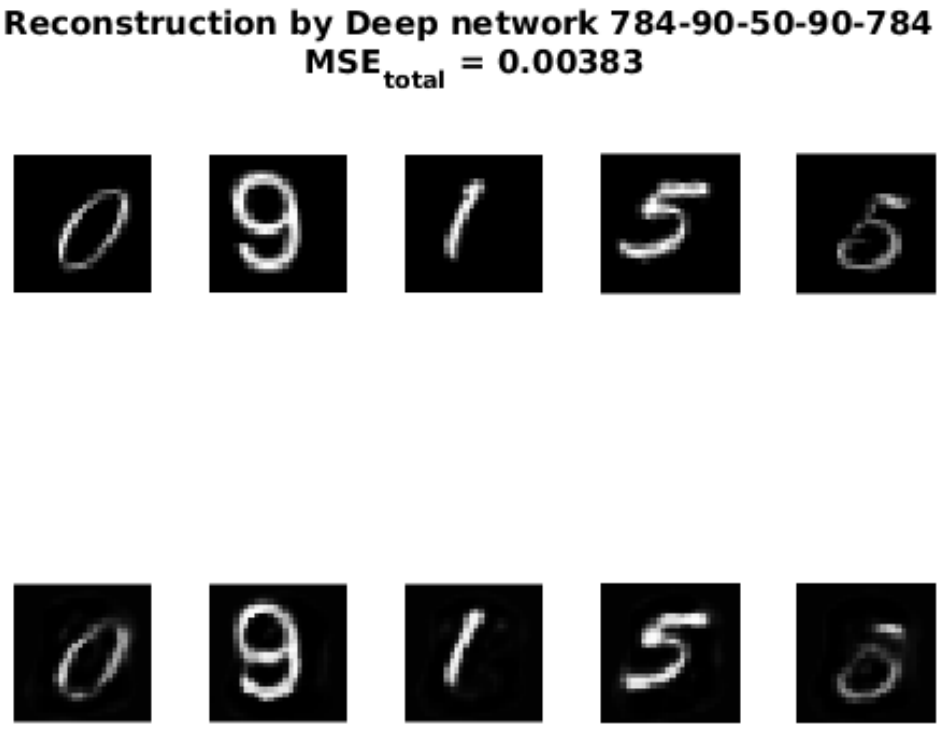


Figure 3: *Reconstruction by 784-90-50-90-784 after PCA initialization*

Dataset	Mean Squared Error
digittrain_dataset	0.00271
digittest_dataset	0.00383

Table 3: *Reconstruction MSE on the whole training and test sets*

1.2 Stacked Ordinary Autoencoder Initialization

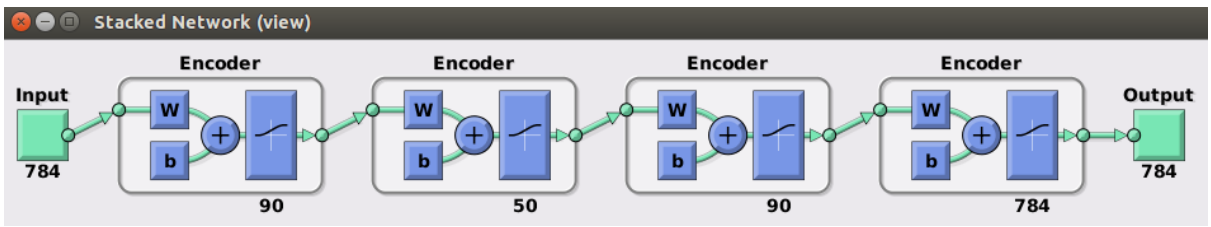


Figure 4: *Deep Neural Network 784-90-50-90-784 with ordinary autoencoder initialization*

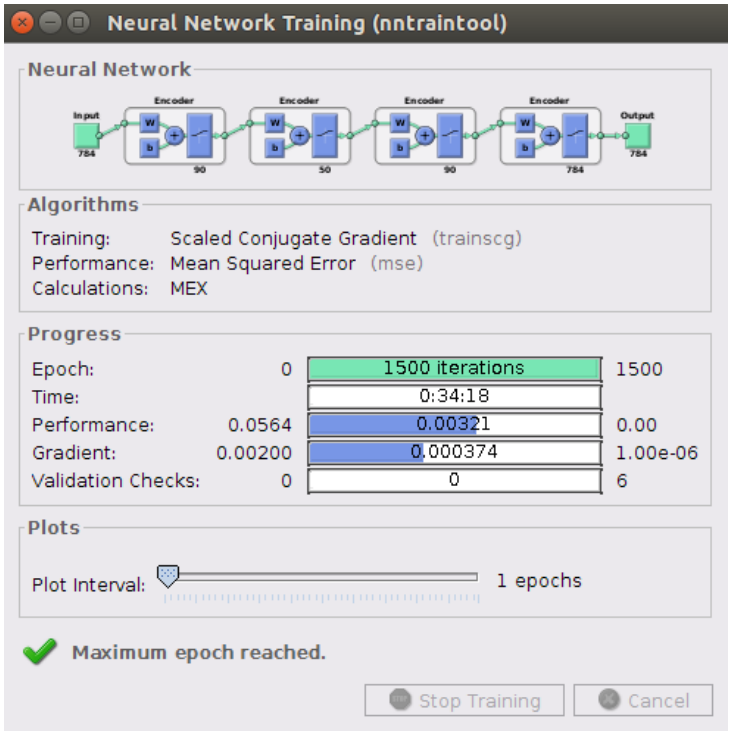


Figure 5: *Fine Tuning training window*

Time (hrs-min-sec)	
Pre-training	00:50:24
Fine Tuning	00:34:18
Total	01:24:42

Table 4: *Time requirements on local machine*

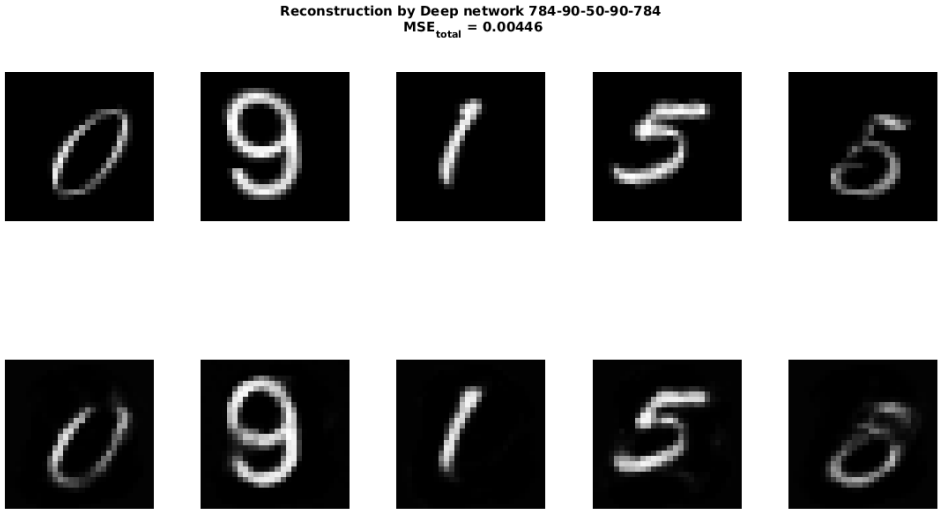


Figure 6: *Reconstruction by 784-90-50-90-784 after ordinary autoencoder initialization*

Dataset	**M**ean **S**quared **E**rror
digittrain_dataset	0.00321
digittest_dataset	0.00446

Table 5: *Reconstruction MSE on the whole training and test sets*

1.3 Stacked Denoising Autoencoder Initialization

A *denoising autoencoder* is like an *ordinary autoencoder*, with the difference that during learning, the input seen by the autoencoder is not the raw input, but a stochastically corrupted version of it. Thus, a denoising autoencoder is trained to reconstruct the input from the noisy version. The basic concept is to force the hidden layer to discover more robust features and prevent it from simply memorizing the identity.

The corrupted version of input data can be obtained by randomly disable some of the input data. For instance, with images as inputs, like in our case, this could be achieved by randomly setting some of the pixels to zero (pepper noise).

As far as the *stacked denoising autoencoders* are concerned, each one of them is trained separately on a corrupted version of the input from the previous layer. After the completion of this pre-training procedure, the deep neural network is fine-tuned using the uncorrupted (original) input data.

For my part, I did not make use of an existent implementation or package concerning the denoising autoencoders. In fact I used the same approach applied to ordinary stacked autoencoders, with the difference that the input pixels at each layer are randomly corrupted by pepper noise, whose density is a user-defined parameter; in my case, it is set to $d = 0.1$. These corrupted data are only used during the pre-training process, while the original data are presented during the global training process.

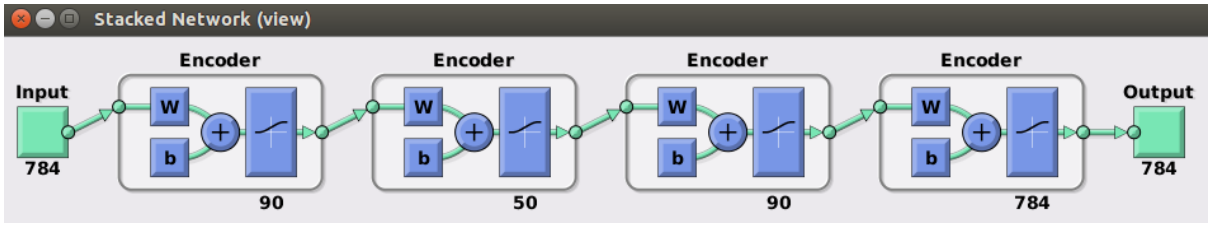


Figure 7: *Deep Neural Network 784-90-50-90-784 with denoising autoencoder initialization*

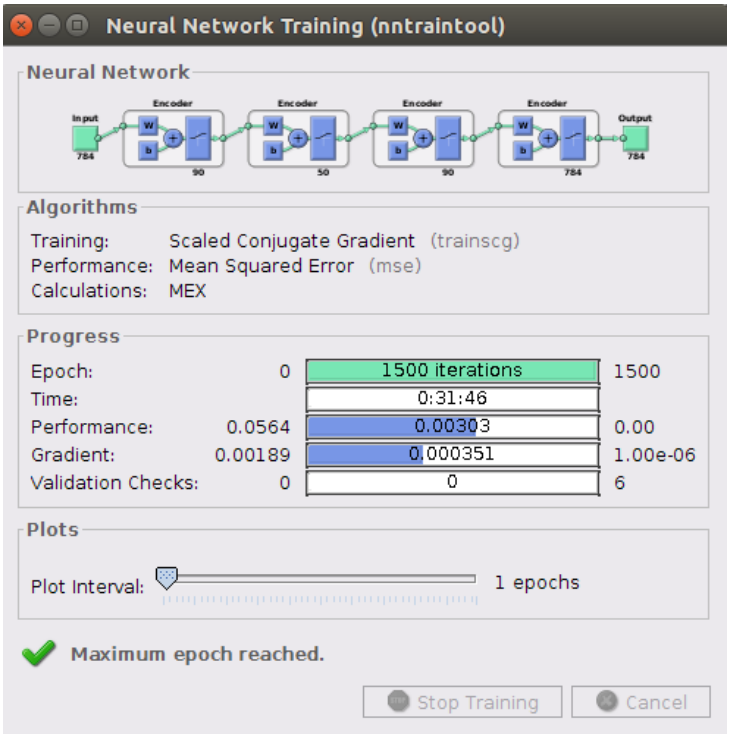


Figure 8: *Fine Tuning training window*

Time (hrs-min-sec)	
Pre-training	00:50:12
Fine Tuning	00:31:46
Total	01:21:58

Table 6: *Time requirements on local machine*

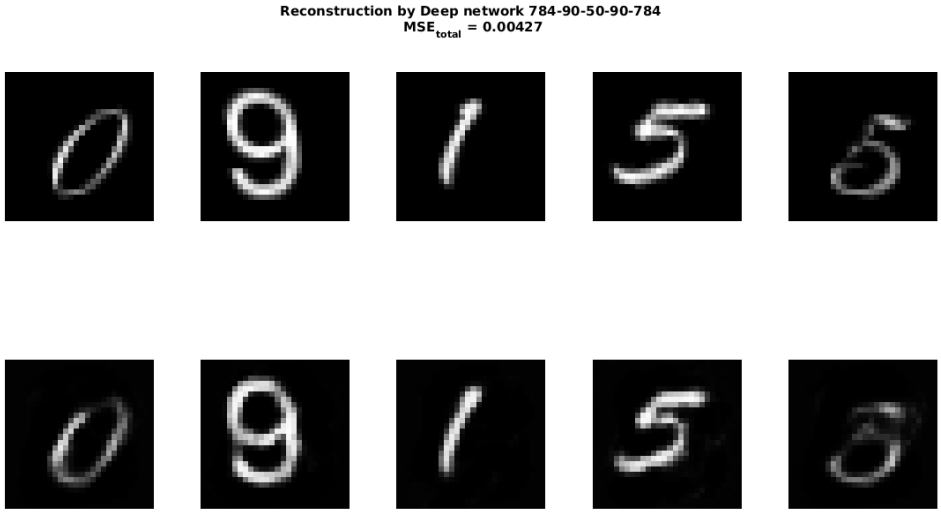


Figure 9: *Reconstruction by 784-90-50-90-784 after denoising autoencoder initialization*

Dataset	**M**ean **S**quared **E**rror
digittrain_dataset	0.00302
digittest_dataset	0.00427

Table 7: *Reconstruction MSE on the whole training and test sets*

The following table summarizes the results obtained by all the *three* different employed methods of deep neural networks' initialization, as well as by applying *plain PCA* (compression from 784 to 50 dimensions and reconstruction) from the previous assignment (Exercise 3):

Method		**M**ean **S**quared **E**rror
-	Plain PCA ($M = 50$)	0.00642
Initialization	PCA	0.00383
	SAE[1]	0.00446
	DSAE[2]	0.00427

Table 8: *Reconstruction MSE on the independent test set*

[1] Stacked Autoencoders
[2] Denoising Stacked Autoencoders

2 Task B

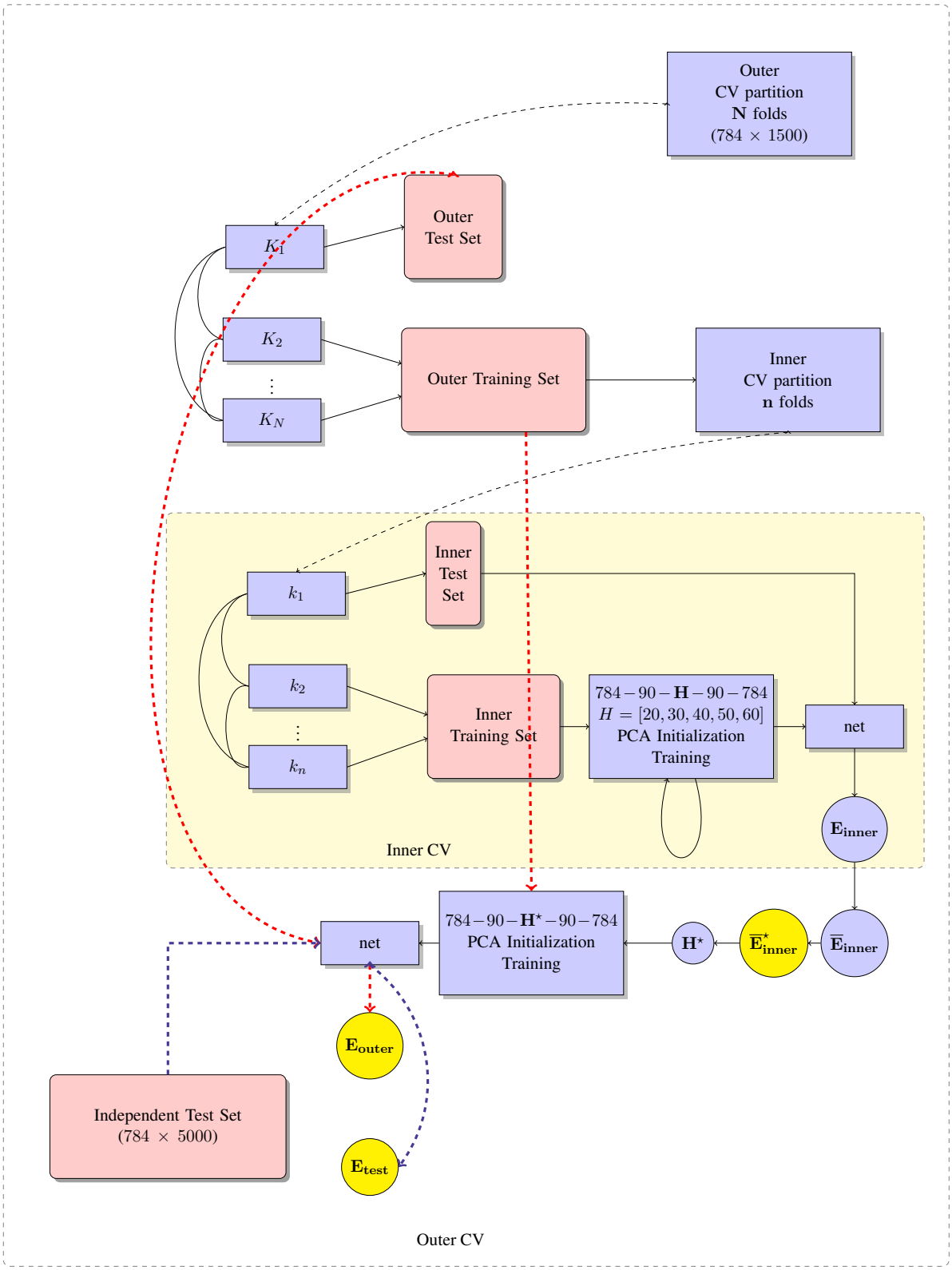


Figure 10: *Flow chart of the double cross-validation procedure*

Figure (10) illustrates the double cross validation procedure that optimizes the number of neurons H in the second hidden layer with limited risk of overfitting.

At this point, it is important to mention that it is proposed to use 600 images of the digittrain_dataset, half of them from the class "3" and the other half from the class "other". This means that the input matrix of vectors of images would be 784×600, resulting in less observations than dimensions. Applying PCA in this dataset, gives at most $600 - 1 = 599$ non-zero eigenvectors. In other words, the rank of the coefficient matrix among the 784 dimensions is 599.

However, the coefficient matrix obtained by PCA is a square matrix $p \times p$, where p is the number of dimensions; 784 in this case. Consequently, in order to avoid the resulting errors, I chose to increase the number of examples to 1600, keeping in mind the consecutive further permutations, so that always an input dataset of at least equal number of dimensions and observations is obtained, both in outer and inner cross validation loops. The reduced input dataset with 1600 images that is used in task B consists of 493 "3" examples, which is the total number of "3" digit images in the digittrain_dataset, and 1107 "other" examples. It should be kept in mind that by decreasing the number of the outer cross validation folds, the size of the input subset should be increased, in order to avoid problems with PCA, as already mentioned.

Finally, I would like to state that I performed some smaller tests regarding the inner cross validation procedure, in order to examine the impact of the bottleneck's size in the respective inner error estimate; E_{inner} in figure (10). I realized that the largest number of H, among all inner iterations, was accompanied with the minimum inner error. For this reason and provided that the computations were heavy, I decided to cut down the size of H and restrict it to take values in the set $[50, 60]$. The time required for one double cross validation with 10 outer folds, 3 inner folds and $H = [20, 30, 40, 50, 60]$ was 17 hours and 23 minutes. Thus, it was necessary to reduce the computational requirements in order to repeat the procedure for more results.

2.1 *One* Double Cross Validation

In this part, **one double cross validation** procedure was performed. More precisely, the number of the outer and inner folds were set to **5** and **3**, respectively. The obtained error estimates, which are annotated as yellow circles in figure (10), are summarized in table (9):

	Mean	Standard Deviation
$\overline{E}^{\star}_{inner}$	0.00811	$3.58e-04$
E_{outer}	0.00561	$3.19e-04$
E_{test}	0.00621	$3.34e-04$
Time	01:54:10	

***Table 9: One** Double CV with K = 5 (outer folds), k = 3 (inner folds)*

2.2 *Three* Double Cross Validations

In this case, the aforementioned double cross validation procedure is repeated **three** times by reshuffling the ordering of the initial outer cross validation partitions of the reduced input dataset. In this way, 3 times more error estimates are obtained for averaging. The obtained results are included in table (10):

	Mean	Standard Deviation
$\overline{E}^{\star}_{inner}$	0.00647	$4.11e-04$
E_{outer}	0.00502	$2.66e-04$
E_{test}	0.00554	$3.02e-04$
Time	06:20:48	

***Table 10: Three** Double CVs with K = 5 (outer folds), k = 3 (inner folds)*

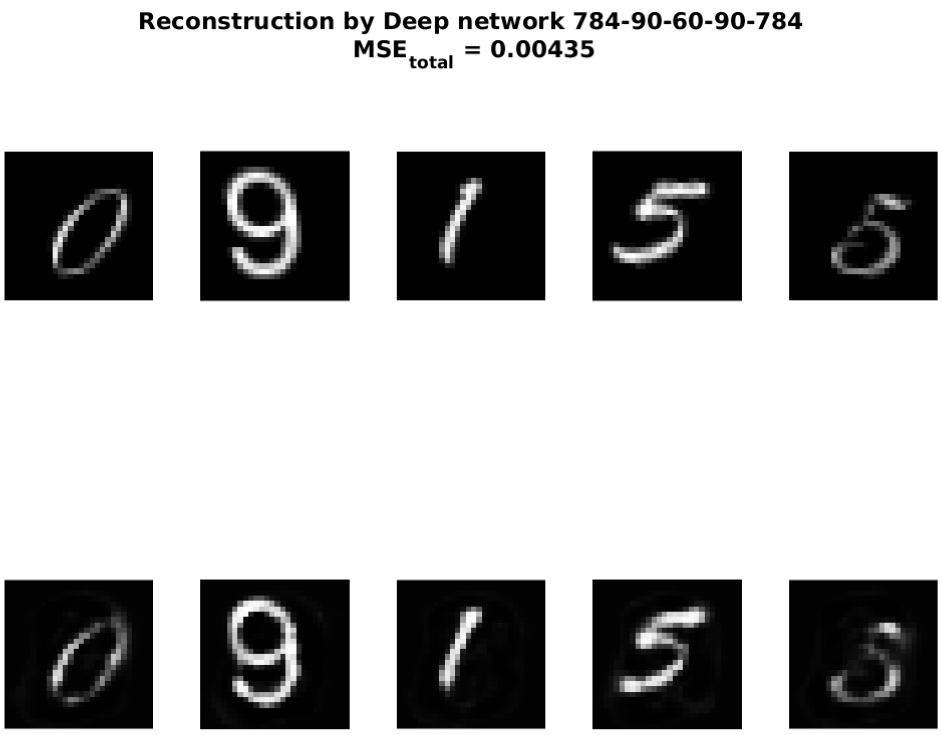
As indicated by both cases, the total inner mean squared error($\overline{E}^{\star}_{inner}$) is larger than the other two. This could be explained by the fact that during the inner cross validation loops the number of training examples are reduced compared to the outer cross validation loops.

As far as the performance in the independent test set with all the 5000 examples (digittest_dataset) is concerned, the average prediction error(E_{test}) is quite close to the outer cross validation(E_{outer}), but still slightly larger. This small difference is expected, since the independent test set contains examples that were not presented during the training of the double cross validation procedures, where a reduced subset (1600 examples) of the digittrain_dataset is used.

2.3 Visual Inspection

Apart from the estimated average error estimates, it is always more convenient to have a visual inspection of the achieved reconstruction. For this reason, the five first original vs. reconstructed images of the independent test set among the three different double cross validations with 5 outer folds are attached below:

1st Double cross-validation among 5 outer folds



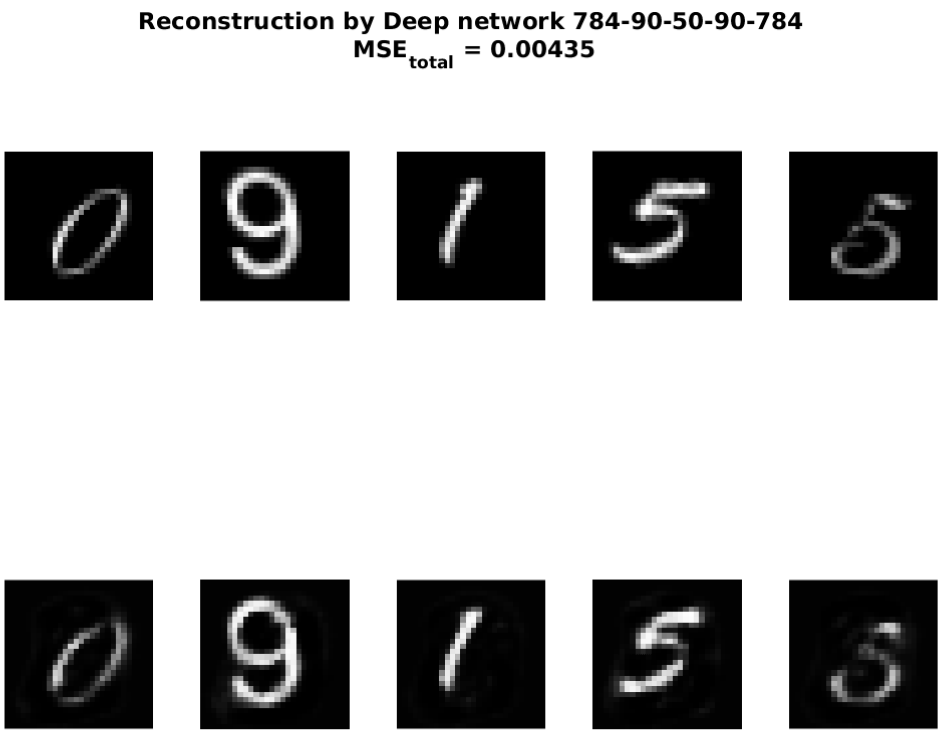
Reconstruction by Deep network 784-90-50-90-784
MSE_{total} = 0.00514

Reconstruction by Deep network 784-90-60-90-784
MSE_{total} = 0.00438

Reconstruction by Deep network 784-90-50-90-784
MSE_{total} = 0.00524

Reconstruction by Deep network 784-90-50-90-784
MSE_{total} = 0.01004

2nd Double cross-validation among 5 outer folds



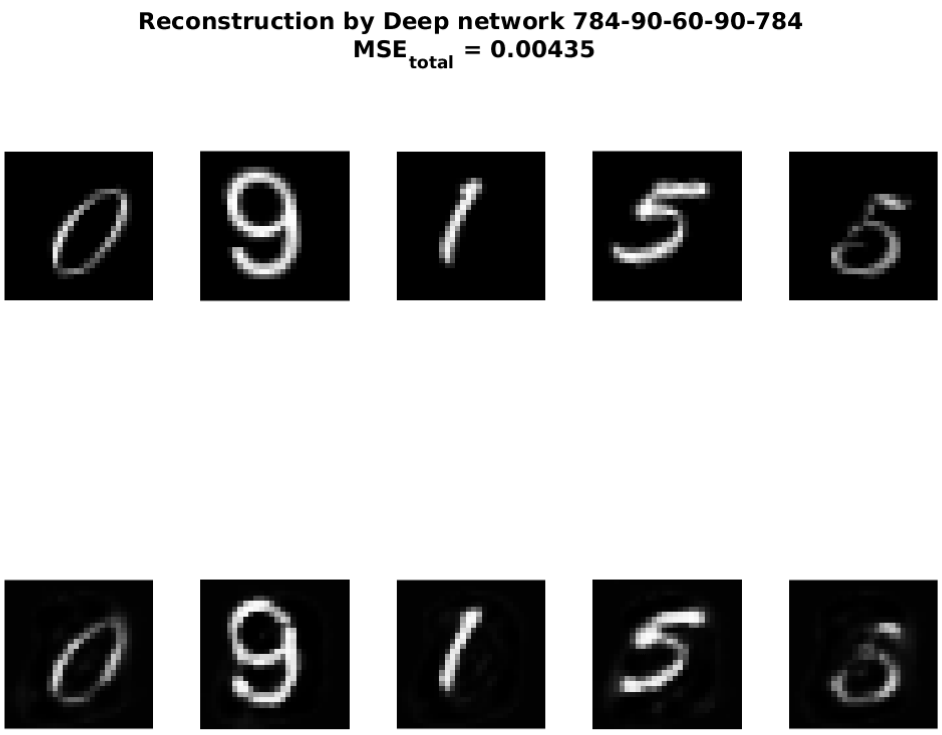
Reconstruction by Deep network 784-90-60-90-784
MSE_{total} = 0.00514

Reconstruction by Deep network 784-90-50-90-784
MSE_{total} = 0.00438

Reconstruction by Deep network 784-90-60-90-784
MSE_{total} = 0.00524

Reconstruction by Deep network 784-90-60-90-784
MSE_{total} = 0.01004

3rd Double cross-validation among 5 outer folds



Reconstruction by Deep network 784-90-60-90-784
MSE_{total} = 0.00514

Reconstruction by Deep network 784-90-50-90-784
MSE_{total} = 0.00438

Reconstruction by Deep network 784-90-60-90-784
MSE_{total} = 0.00524

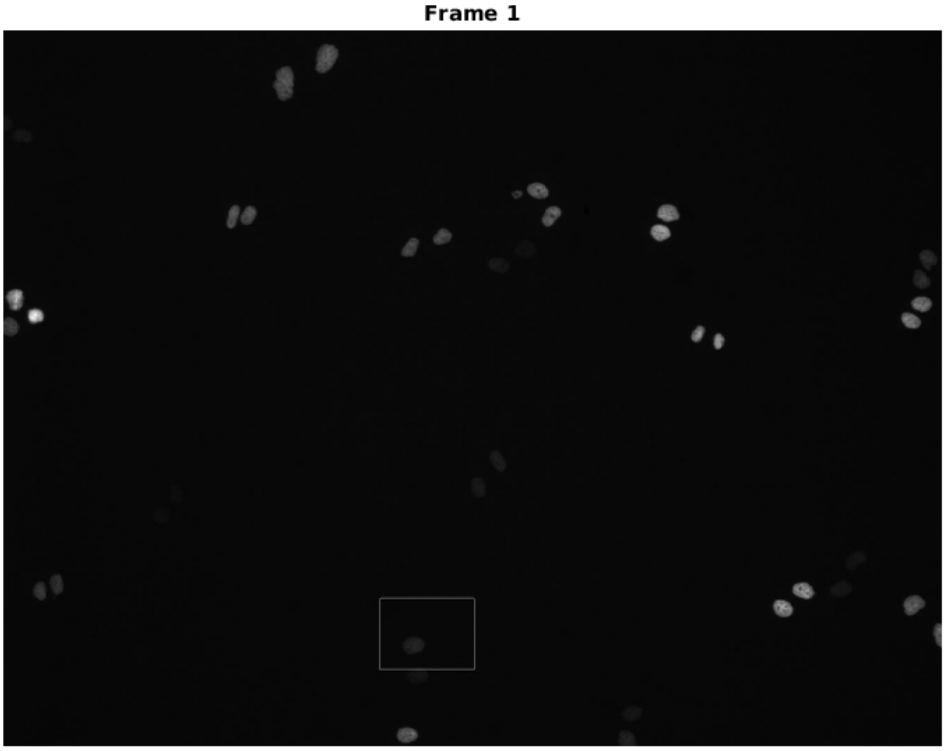
Reconstruction by Deep network 784-90-60-90-784
MSE_{total} = 0.01004

3 Task C

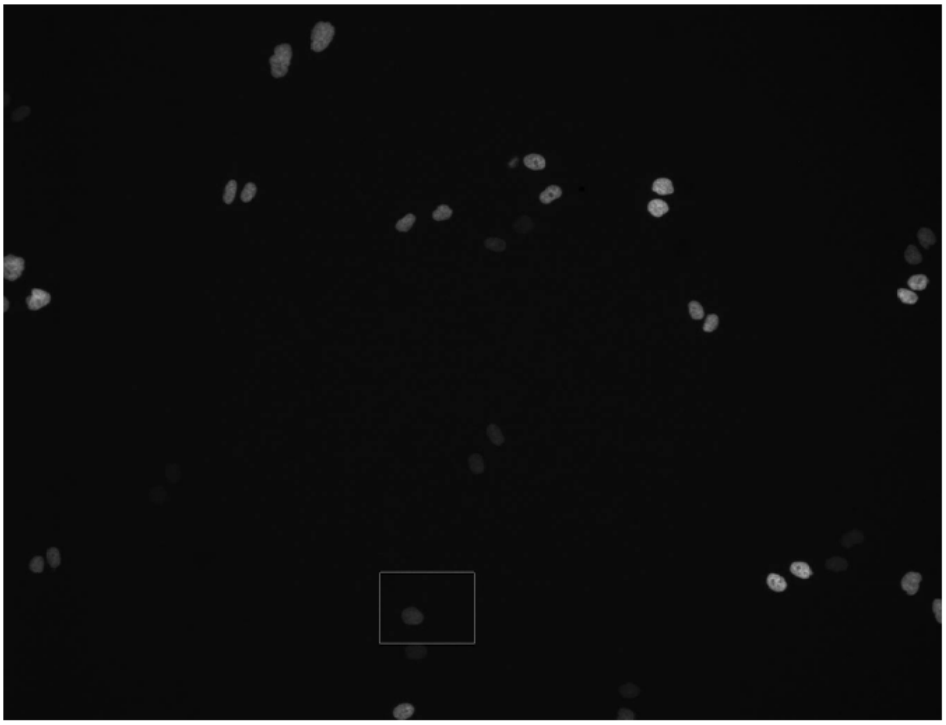
MATLAB R2015b is equipped with the *VideoReader*[2] class, which is capable of creating objects to read video files. The obtained video objects and their properties are then accesible for processing.

The provided time-lapse microscopy movies[3] are in *Apple QuickTime Movie* (.mov) format. Under Linux®, this requires the installation of all the available plug-ins for GStreamer 0.10, as well as an appropriate converter. For me, the installation of *ffmpeg* did work. I would definitely recommend compiling *ffmpeg* from source, since I ran into issues while using the version available through *apt*, possibly due to the fact that Ubuntu no longer uses the official version, but rather a fork.

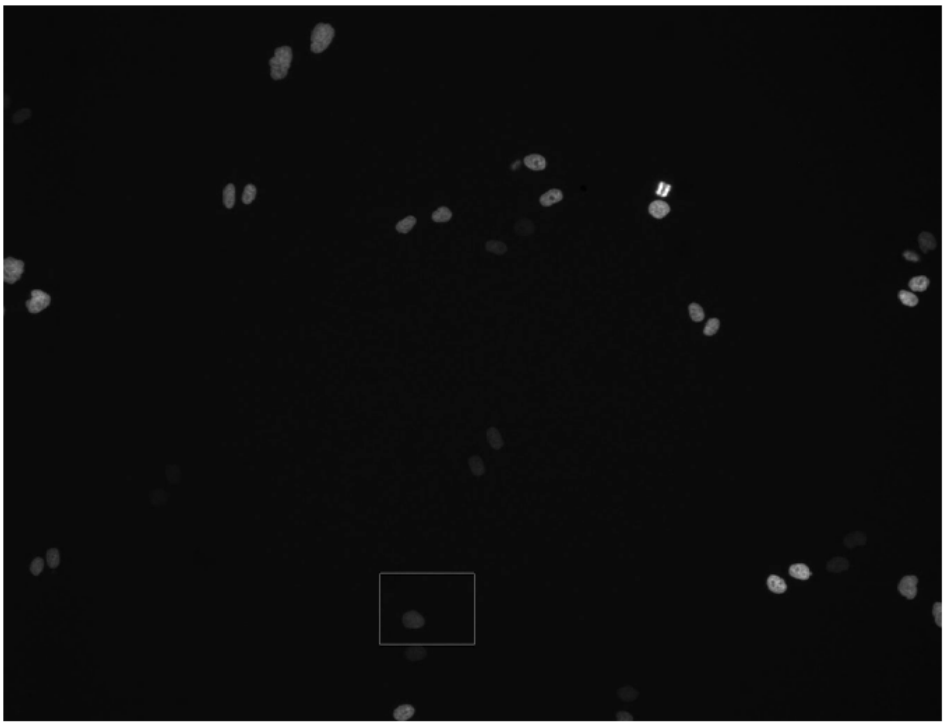
Four different frames of the second time-lapse microscopy movie (2008-09-09955C-Movie2_Elongated interphase nulei due to VIPR RNAi.mov) included in the first set of supplementary movies, are attached below.

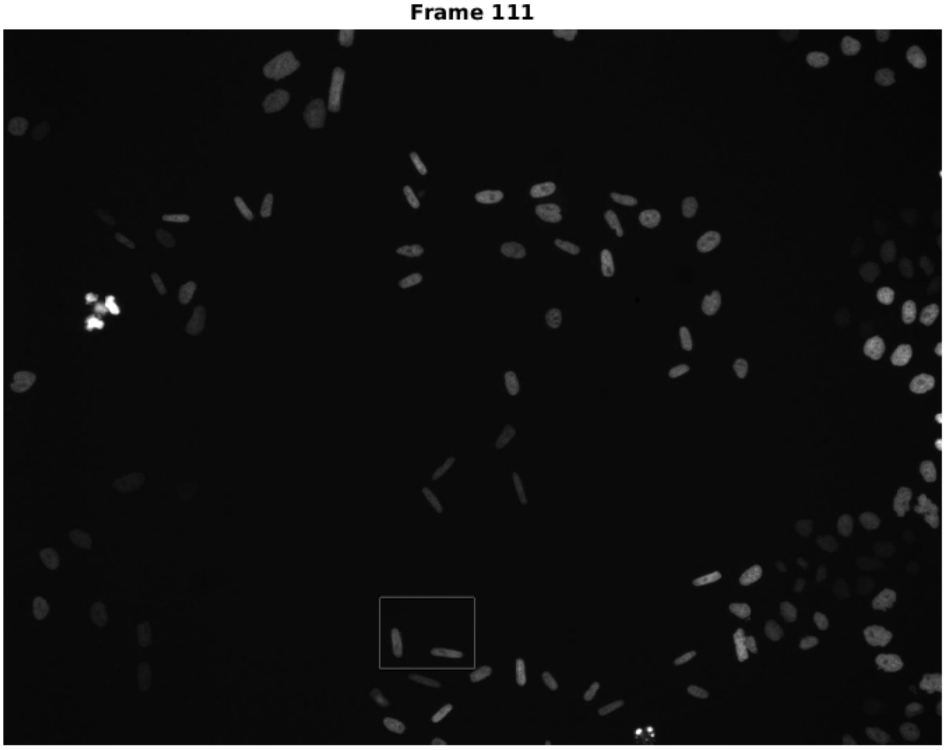


Frame 2



Frame 3




Frame 111

References

[1] "MathWorks Neural Network Toolbox." http://se.mathworks.com/products/neural-network/. Accessed: November 20, 2015.

[2] "VideoReader." http://se.mathworks.com/help/matlab/ref/videoreader.html. Accessed: November 27, 2015.

[3] B. Neumann *et al.*, "Phenotypic profiling of the human genome by time-lapse microscopy reveals cell division genes," *Nature*, vol. 464, pp. 721–727, Apr 2010.

[4] "trainAutoencoder." http://se.mathworks.com/help/nnet/ref/trainautoencoder.html#responsive_offcanvas. Accessed: November 21, 2015.