**Maman 12 report**

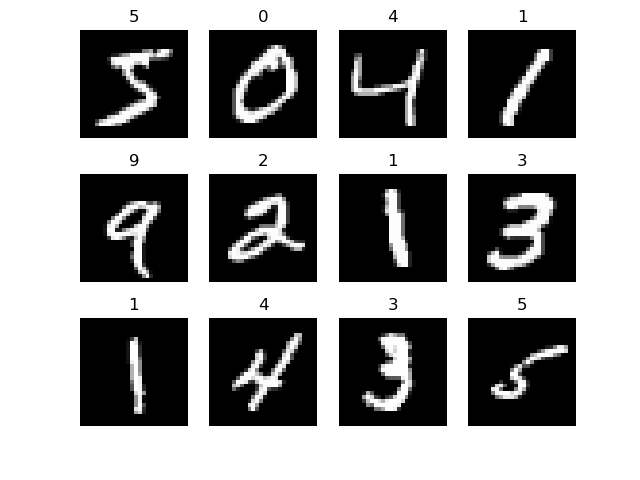
**Noam Salomonski**

**303161194**

Loader.DataSet loads the database from pkl, and shows the required visualizations.

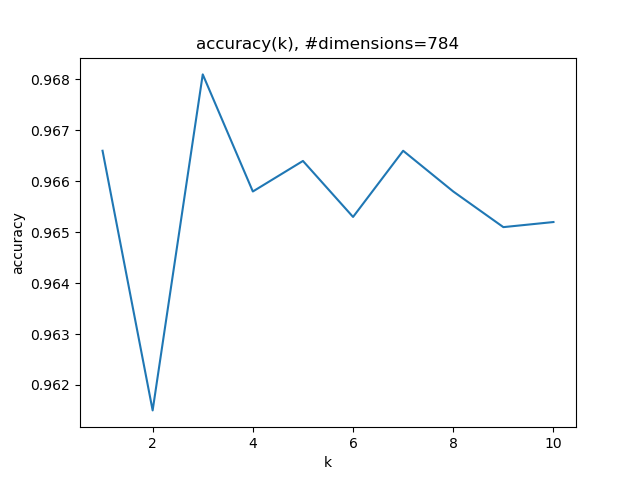
Counting occurrences of every digit (in the y vectors) in all train/validation/test sets gives the following histogram:

{0: 6903, 1: 7877, 2: 6990, 3: 7141, 4: 6824, 5: 6313, 6: 6876, 7: 7293, 8: 6825, 9: 6958}

Showing 

Question 1 – KNN:

Showing the results for k=1..10:



We can see the accuracy for k=3 is 0.968. [the graph starts from 0, but k from 1]

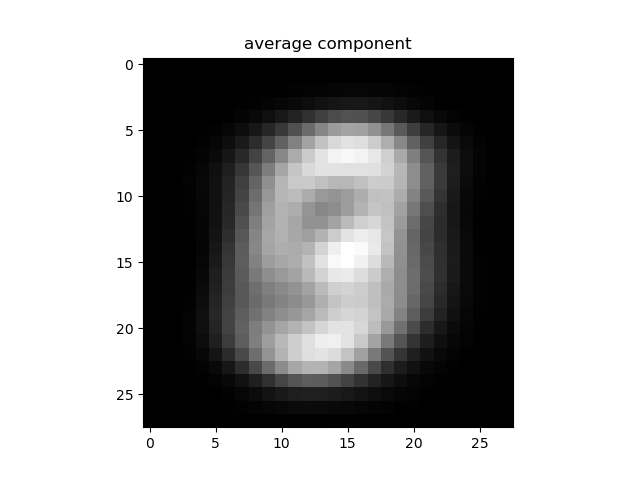
In case of a tie in “votes”, the lower digit is selected.

**Question 2 – PCA:**

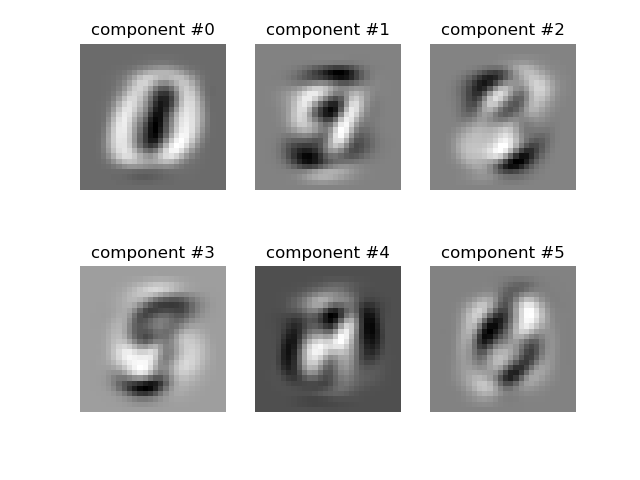
b.

The average digit is the same if calculated in pca or in feature (pixel) space, because of linearity.

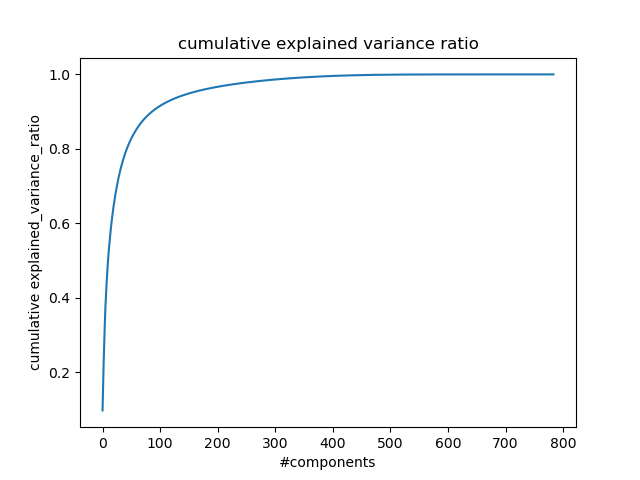
Here it is in feature space:



The first 6 strongest components in feature space:



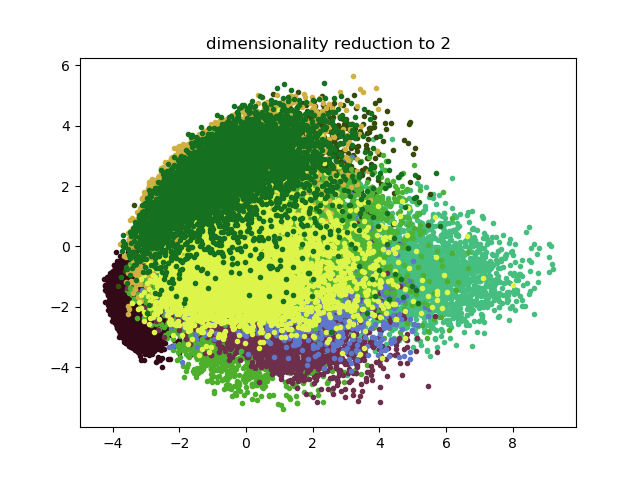
Cumulative variance which is explained by (sorted) principal components (1 is all the variance).



If we want to preserve 80% of the variance, we need the first 44 components. For 95%, we need 154 components.

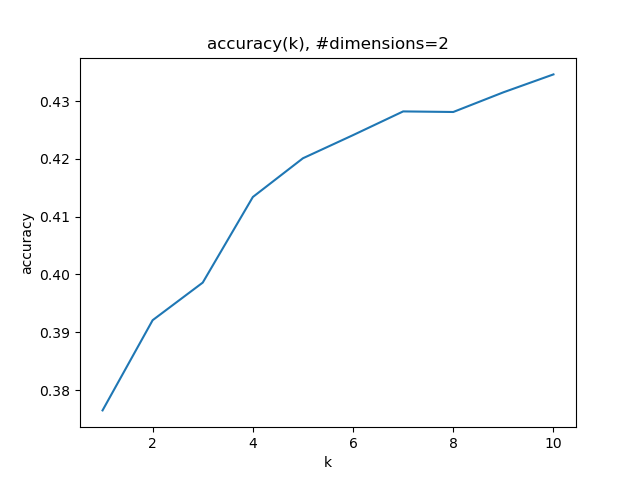
Calculated using calculate\_n\_components\_required.

Reducing the dimensionality to 2 by using the strongest 2 components and giving each label its own color yields the following:



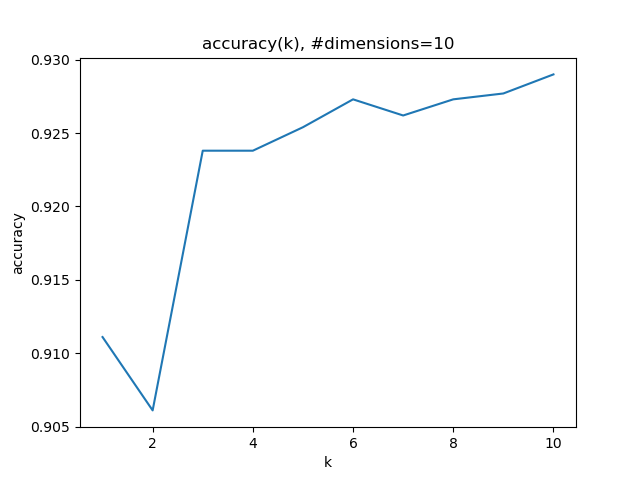
f.

When repeating question 1 with 2 dimensions:



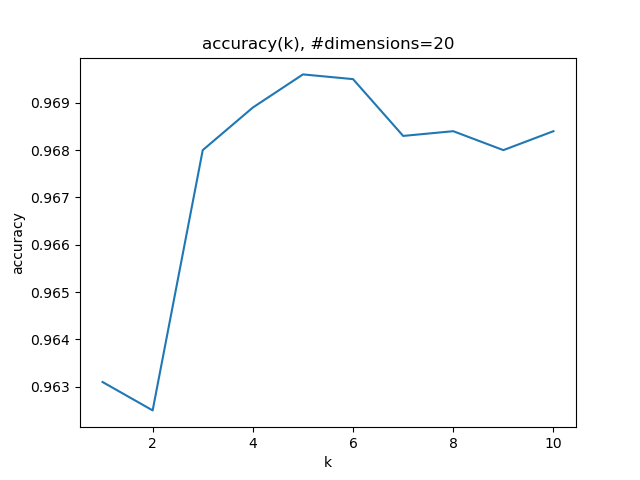
Best k is 10. Accuracy is very low, because there are not enough dimensions. Computation is noticeably faster.

With 10 dimensions:



Best k is 10. Better accuracy, but not as good as before.

With 20 dimensions



Best k is 5, and accuracy is even better than with all dimensions!

**For transforming a digit to a lower dimension:**

reduced\_components = pca.components\_.T[:, :dim]  
reduced\_data = np.dot(data - pca.mean\_, reduced\_components)

meaning: reduce the mean from the digit’s data in feature space, then matrix-multiply by the first dim pca vectors.

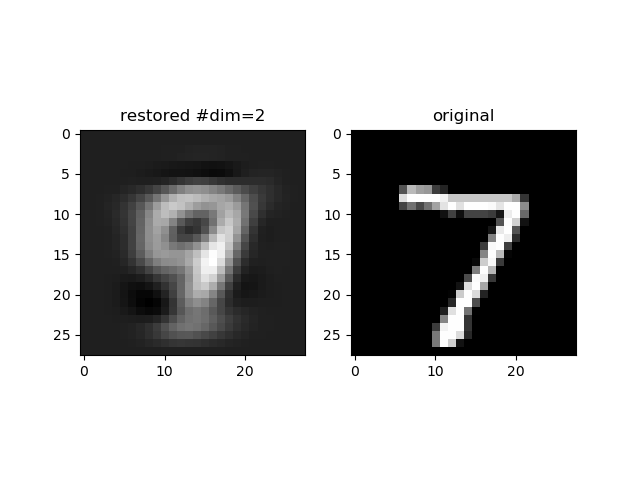
**For transforming a digit back from pca space:**

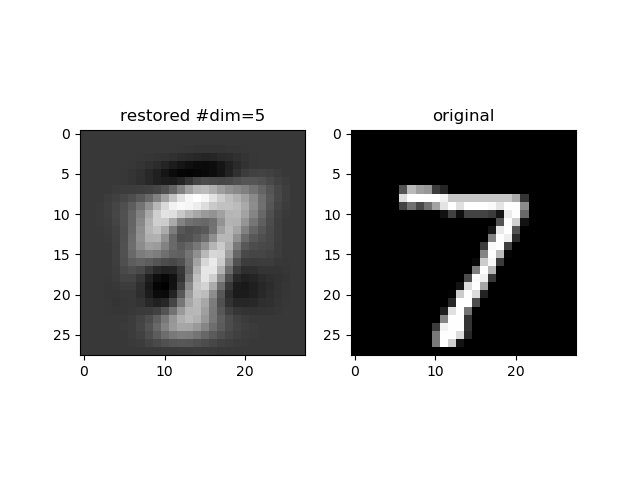
data\_original = np.dot(pc\_data, pca.components\_[:dim, :]) + pca.mean\_

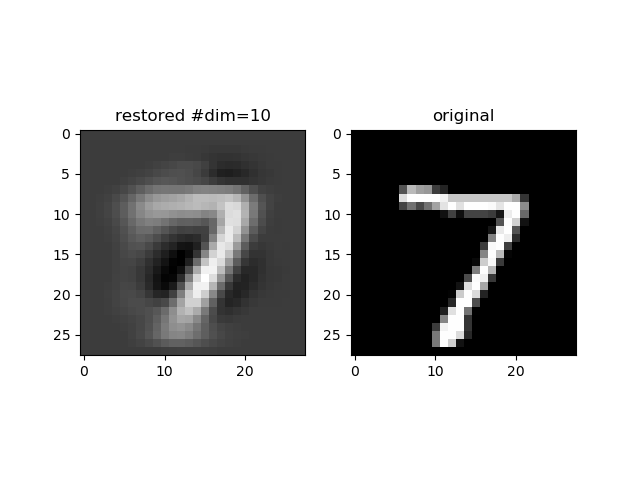
meaning do the exact opposite:

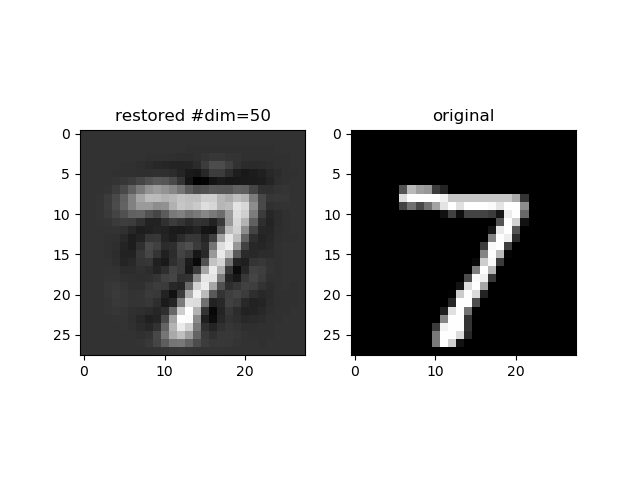
multiply by the 1st dim components (transposed == inverse), then add the mean.

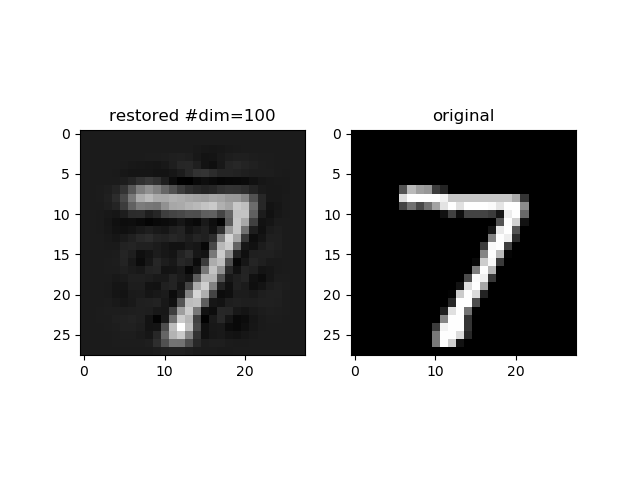
Results for different dimensions:

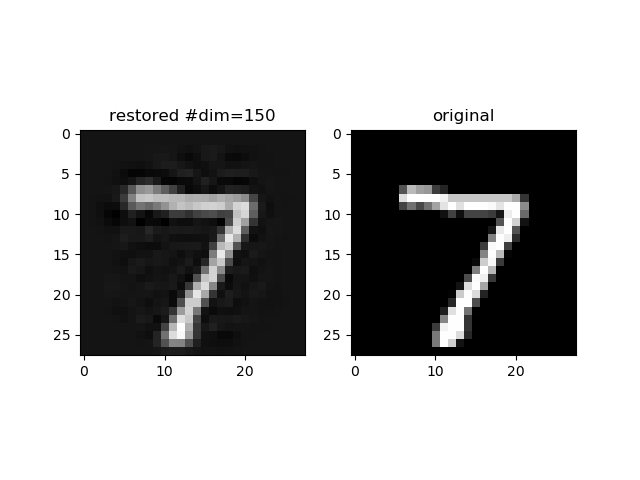








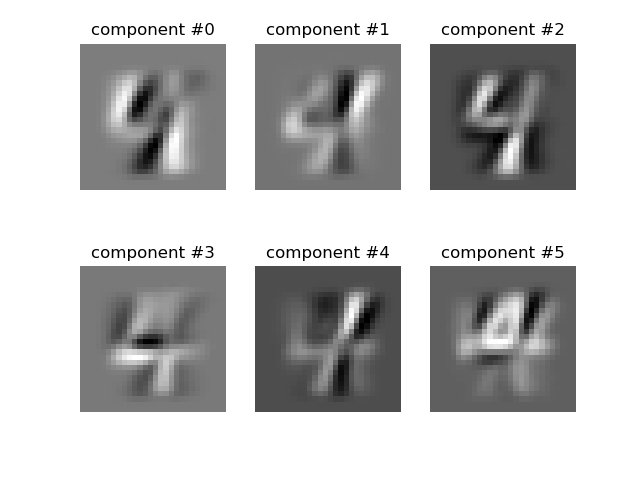
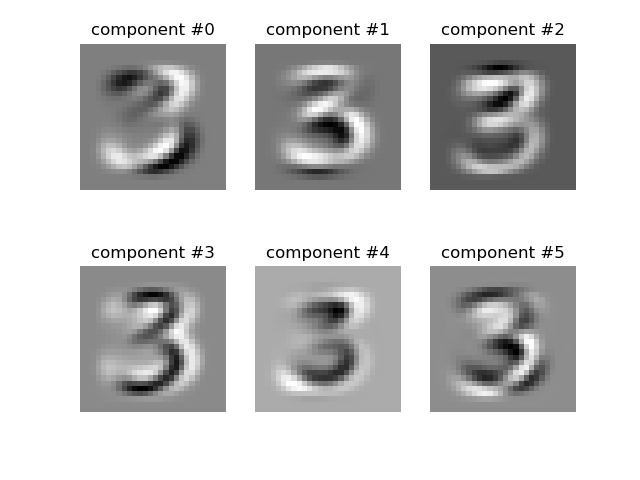
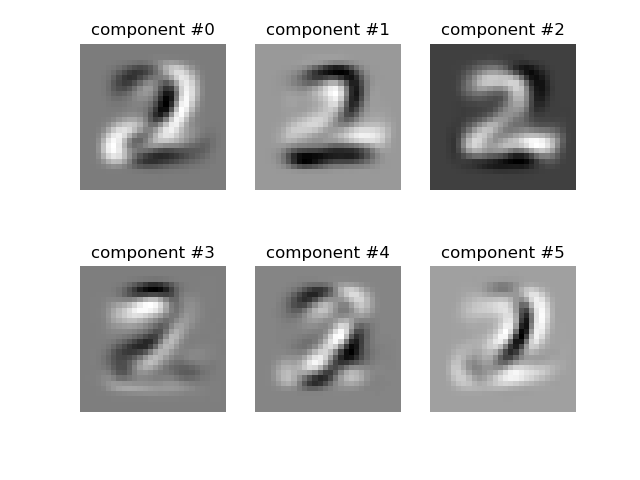
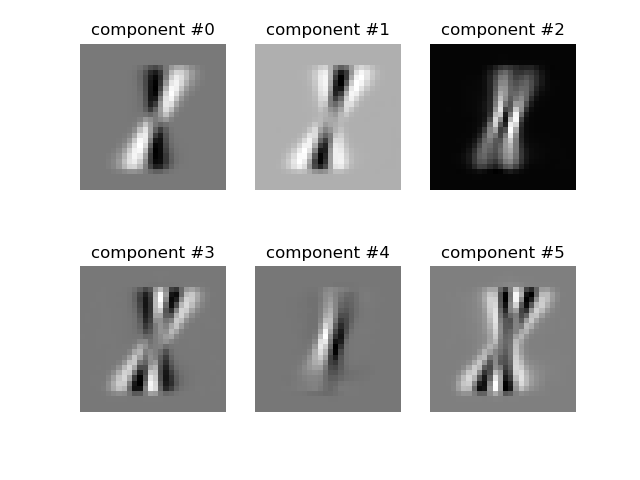
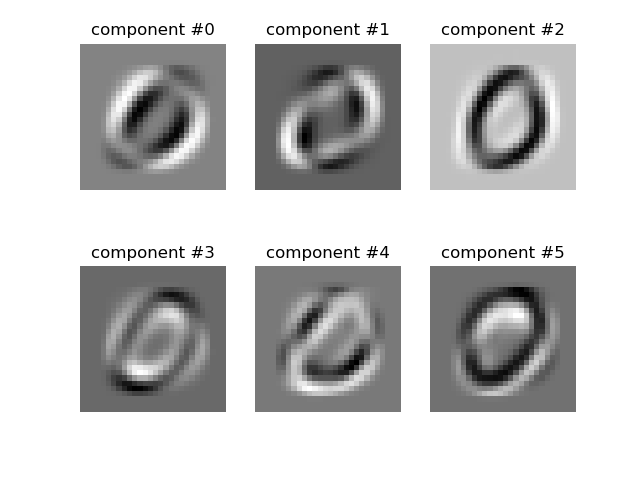
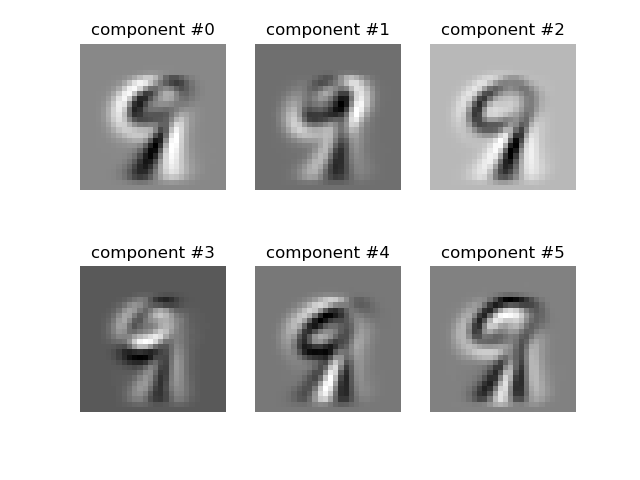
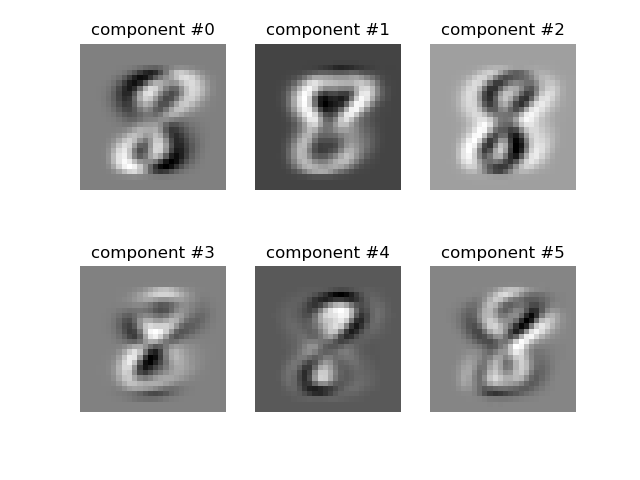
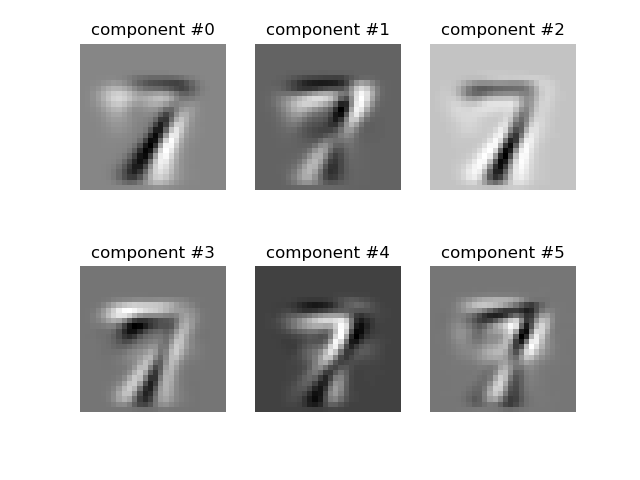
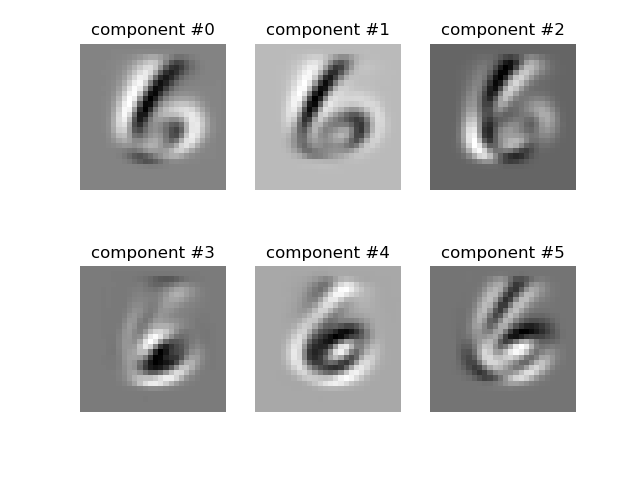
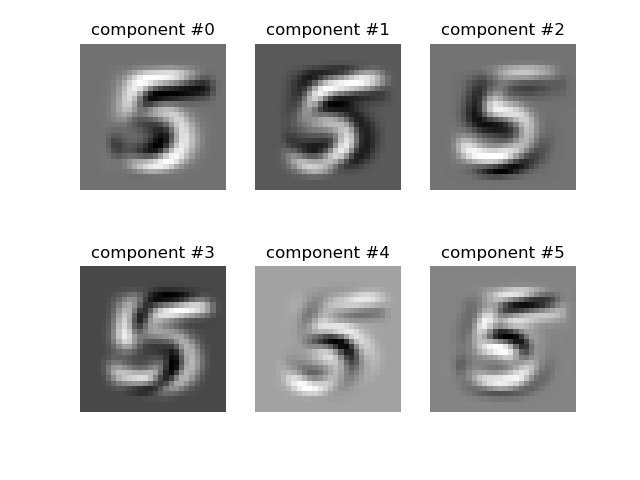




We can indeed see that the more dimensions we keep, the better the reconstruction gets.

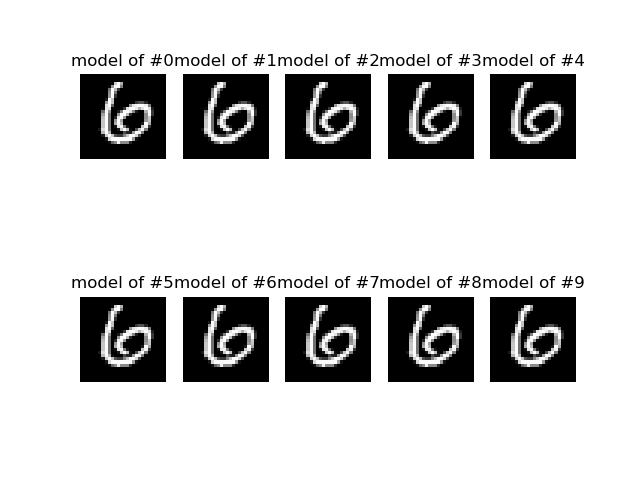
h.

If we use a separate pca model for each class, we get much clearer results for the main components:



It is clear which model belongs to which digit. This happens because the model does not have to account for more than one digit, and the variance within the digit’s pictures is much smaller, thus the model can account for “inner variance” or “types of 4”, rather than “types of digits”.

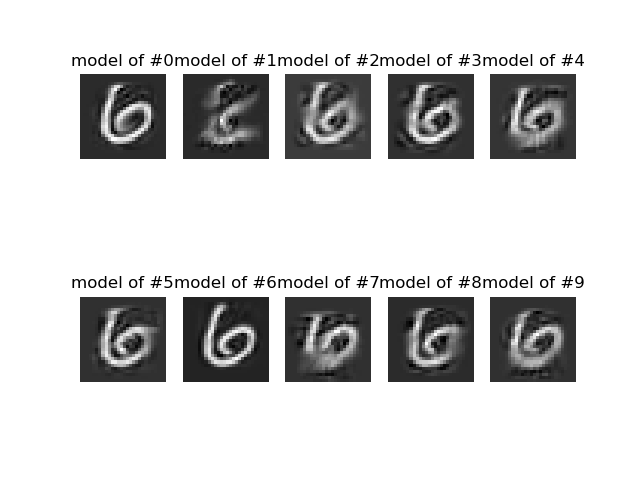
h.



Using different models for each digit, projecting by the model and restoring gives images that seem very similar. The above shows a digit which seems very similar to the original.

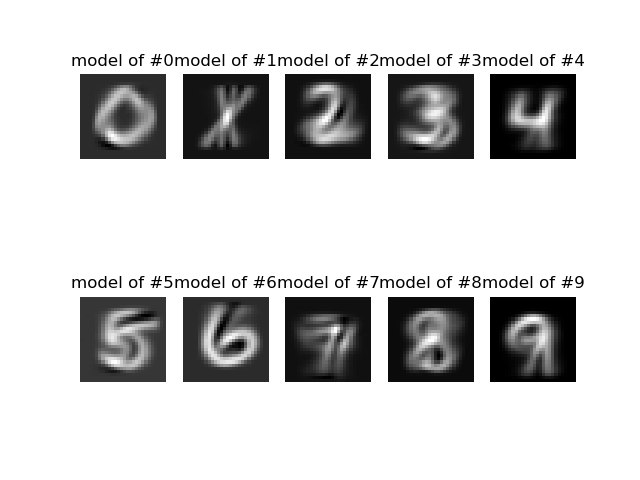
The performance of such an algorithm on the test set is an accuracy rate of 0.8086, which can be explained by numeric error. All the differences between the restored images and the original are about 1e-11.

When reducing the dimension to 100 and restoring, we get



And indeed the performance improves to 0.9291 accuracy.

And even better with only 6 dimensions



Accuracy: 0.9375.

This can be explained by the original being much more likely to be closer to “an average digit” that came from its distribution than other distributions.