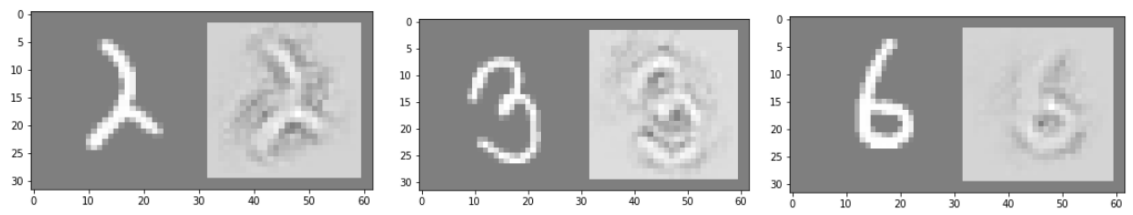
Visualizing and understanding convolutional networks

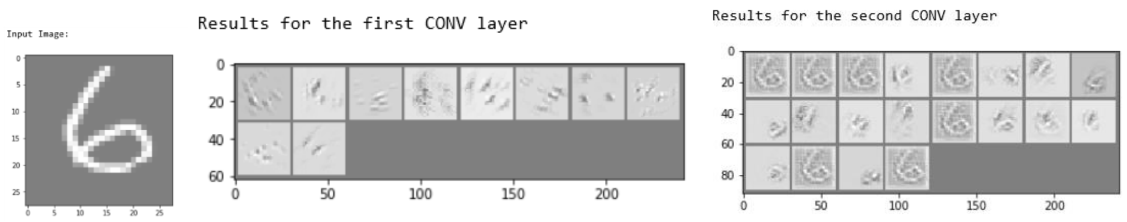
Paper | Notes

Implementation | Implementation II | Implementation III

A **deconvolutional network** can be thought as a reverse CNN, in which features are being mapped back into the image space. The network has an architecture which is a mirror image of the original model with the main components (e.g. pooling, convolutions, non-linearities) reversed. **Max Pooling** can be partially reverted, if the indices from the original CNN are saved and loaded in the **DeConvNet**. **Filtering** is reversed by reusing a transposed version of the learned weights of the original model. If we want to showcase a single activation, we need to set all other filters from the same layer to zero in order to examine the importance of the chosen filter.
DeConvNet can be used for unsupervised learning, as well.

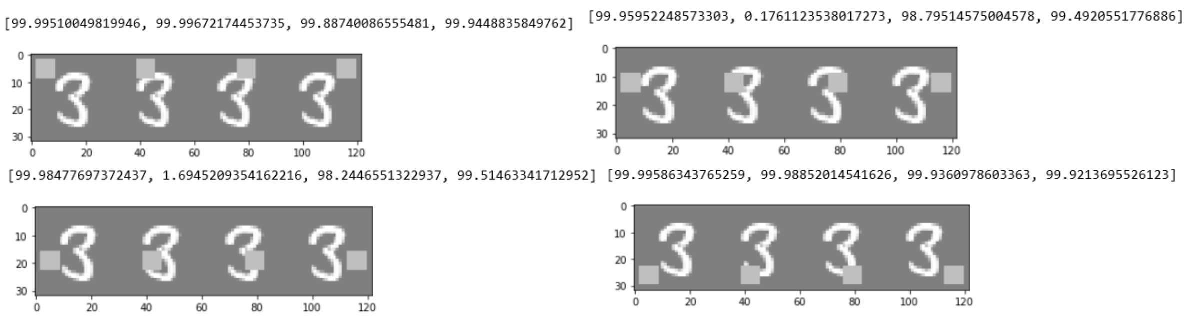


Results: The results from the **DeConvNet** model for the numbers 2, 3 and 6 by using all feature maps.



Results: Input and feature maps activations for each filter for the two CONV layers in the model.

Occlusion Sensitivity refers to the technique of monitoring the output of a classifier by hidding parts of the original image. Based on that, the regions of the image can be found, which have the highest importance to the final decision of the model. An example for a picture of the number 3 can be seen. The experiment shows that hiding the central regions of the image leads to plummeting of the probability assigned to the correct class. Based on that, the conclusion can be made, that the classifier correctly locates the position of the object.

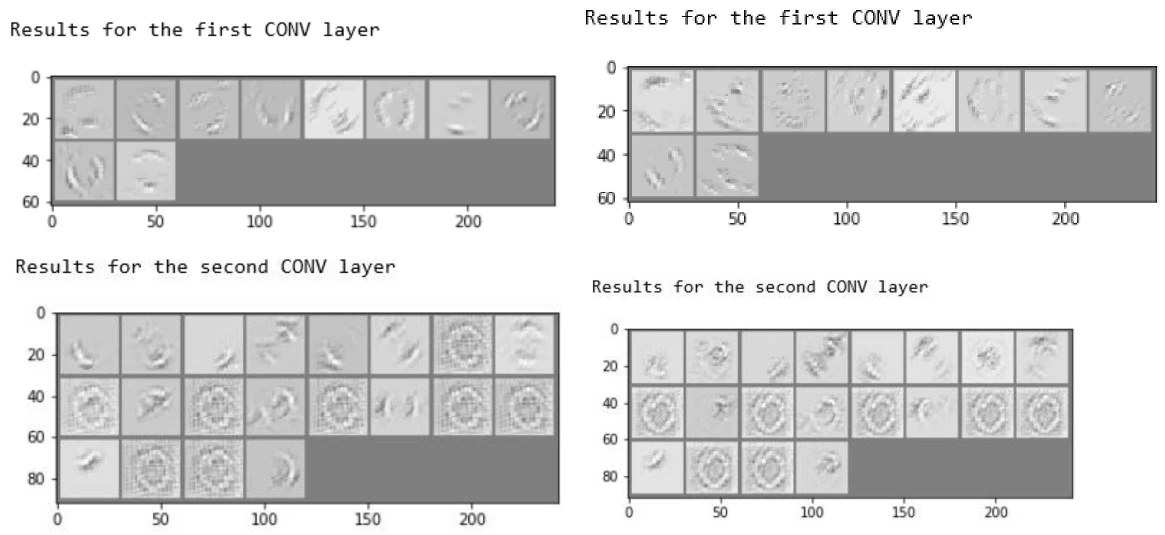


```
[99.99510049819946, 99.99672174453735, 99.88740086555481, 99.9448835849762]
[99.95952248573303, 0.1761123538017273, 98.79514575004578, 99.4920551776886]
[99.98477697372437, 1.6945209354162216, 98.2446551322937, 99.51463341712952]
[99.99586343765259, 99.98852014541626, 99.9360978603363, 99.9213695526123]
```

Results: **Occlusion Sensitivity** - Numbers above pictures are the probabilities which the trained model assigns for each image for class 3.

Feature Evolution during Training

Another possibility to visualize the features that a model learns is by incrementally showcasing them during the training phase. This is achieved by using the learned weights to compute the feature maps activations for a chosen base image after each epochs. Results with these technique can be observed below.



Results: Comparison between feature activations in DeConvNet right after first and fifth epoch of training for a base image. On the left side, the results after the first epochs are depicted. The right side corresponds to the state of the model after the fifth epoch.

LRP

We implemented LRP on the same network as previous while changing all the dense layers to convolution but still keeping the learned weights of the original mode. We chose LRP epsilon rule because the naive LRP rule(alpha1 beta0) leads to numerical instability. Also epsilon can use to absorb relevance when the contributions to the activation are weak. Thus giving a sparse and less noisy result.

LRP epsilon rule:

$$R_j = \sum_k \frac{a_j w_{jk}}{\epsilon + \sum_{0,j} a_j w_{jk}} R_k$$

Naive LRP rule:

$$R_j = \sum_k \frac{a_j w_{jk}}{\sum_{0,j} a_j w_{jk}} R_k$$

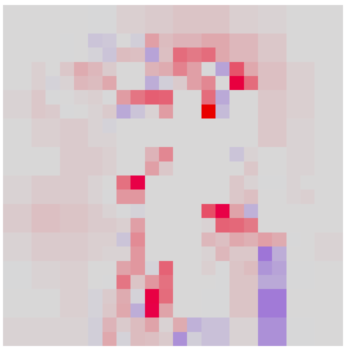
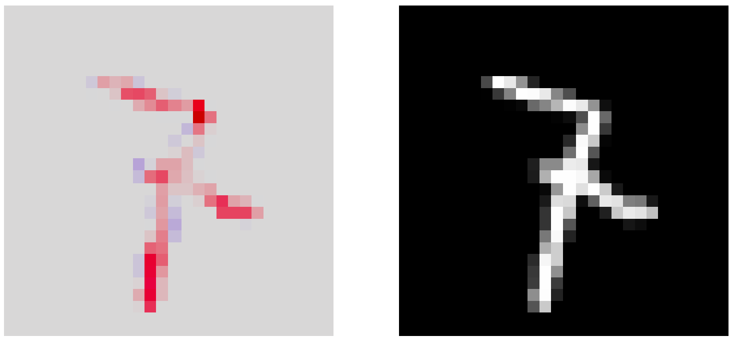


Fig 1 Heatmap for first convolution layer(layer1) Here relevance score are pooled over all feature maps at a given layer


Relevance scores obtained in the pixel layer to indicate the contribution of individual pixels

Q's:

- Is our model overtrained and thus, are somewhat of the result unrealistic?
- Why do the pixels near the numbers have higher relevance scores comparing to the ones inside the numbers (Fig. 1) ? Is it because of the edges?