# **Appendix D: Visualizing Conv Nets**

*Saliency Map*

When a deep convolutional network is trained for classification it should learn to pick up the parts that are distinguishable between classes. In our case we expect that the network will get the information from callosity patterns of whales. To check what parts of images are used in classifier we can do the following:

1. In the current image compute the probability of correct class.
2. Occlude small part of the image, by setting its intensity to 0 (fig.1).
3. Get the output for occluded image from classifier.
4. Compute the difference between the probabilities for the whole image and occluded one. This difference will be the measure of importance of occluded part for classification. The bigger is the difference the bigger is contribution of the region to the final score.
5. Repeat the same procedure for next square box.

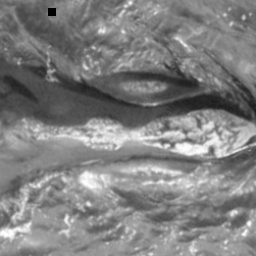
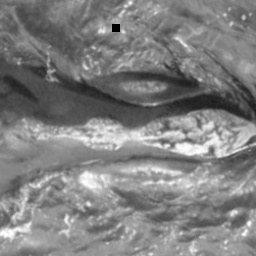
The ‘heat map’ of different images is shown on figure 2. The red pixels correspond to most important parts.

Figure : Occlusion in input image

## w_3.jpgsalence_map_heat.jpgmerged.jpg

Figure : Saliency Map

We can see from salience map that the network is not trained well to distinguish only the callosity pattern. It also takes into account some noise from water and splashes.

## Visualization of filters

## CI_report_filters.png

Figure 3: Output of convolutional layers

As long as the networks use shared parameters for each depth layer of neurons the output of each layer will consist of n filtered images, where n is the depth of the layer. We use these filtered images to understand how the network is actually processing the input. The example of an activation output for each layer is shown in figure 3.

We can see that some of the more successful neurons filter out almost all the water noise and leave only the whale callosity pattern.

There is another use of the neuron visualizations. The networks use ReLU activation function which is simply thresholding output at zero. It was found that this type of activation accelerates the convergence and provides much faster computations comparing to sigmoid functions. But it has a problem of ‘dying neurons’ when some of the weights could be updated in such a way that the neuron will never activate with any input again.

The images of output can help to find out these ‘dead’ neurons. If we compare the outputs of the neurons for different input images we can discover which neurons don’t activate (figure 4).

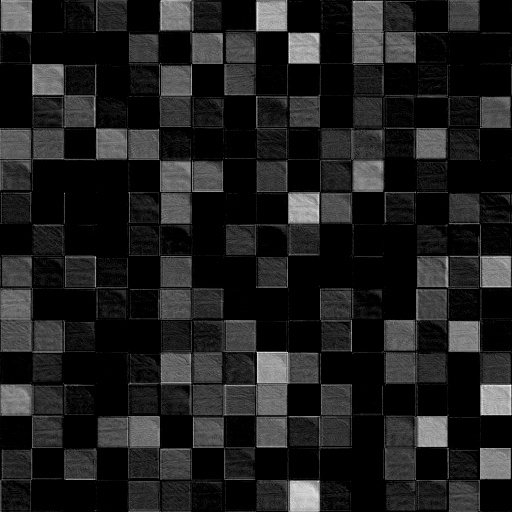
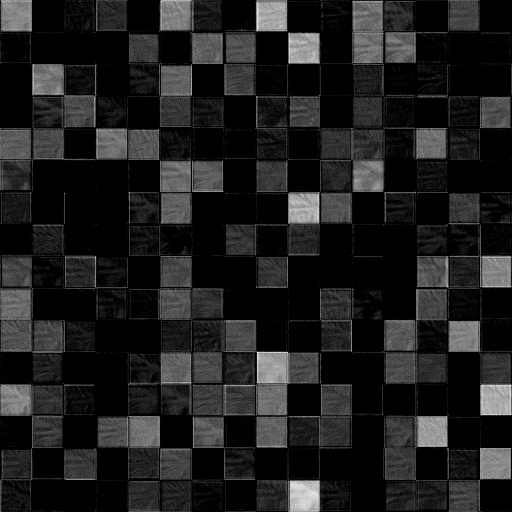
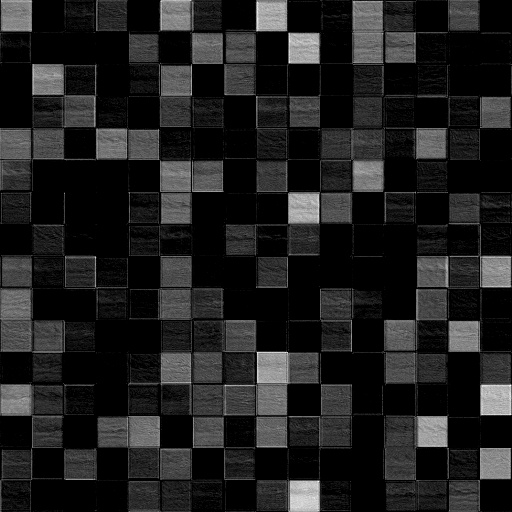


Figure 4: Output from layer 4 for different images

# Appendix E: MNIST Mini Problem

A simple way of illustrating how CNNs perform compared to more naïve methodologies is to show how CNNs work on a simpler problem. The MNIST handwritten digit dataset was chosen for this problem. The dataset consists of 60 thousand training images of 10 classes and 10 thousand of test images. Individual image size is 28x28 pixels.

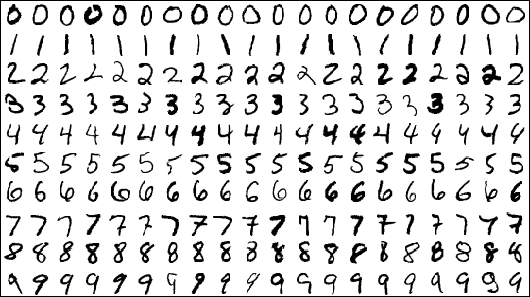


Figure 1: Some MNIST digit images

A network with two convolutional layers was used, each of which was followed by a pooling layer, and one fully connected layer (figure 2). Neuron dropout was applied in the fully connected layer to avoid overfitting.

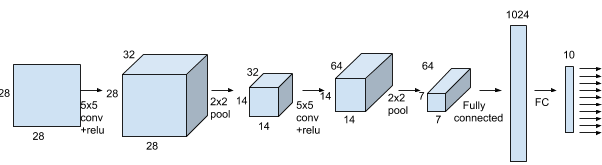


Figure 2: Network architecture for MNIST classifier CNN

A k nearest neighbor classifier is used to compare the performance of the CNN. The k-NN classifier uses the following parameters: k = 5 and Minkowski distance.

It is compared to the neural network (fig. 2) trained with 20000 steps, mini batch size 50, learning rate 10-4 and dropout probability 0.5. Using 10-fold cross validation on the data set we were able to calculate and compare accuracy of kNN vs the CNN implementation.

The accuracy for k-NN was 96.88%. With the CNN we obtain an accuracy of 99.18% which is approaching human error rate.