Neural Networks for Images - Exercise 1

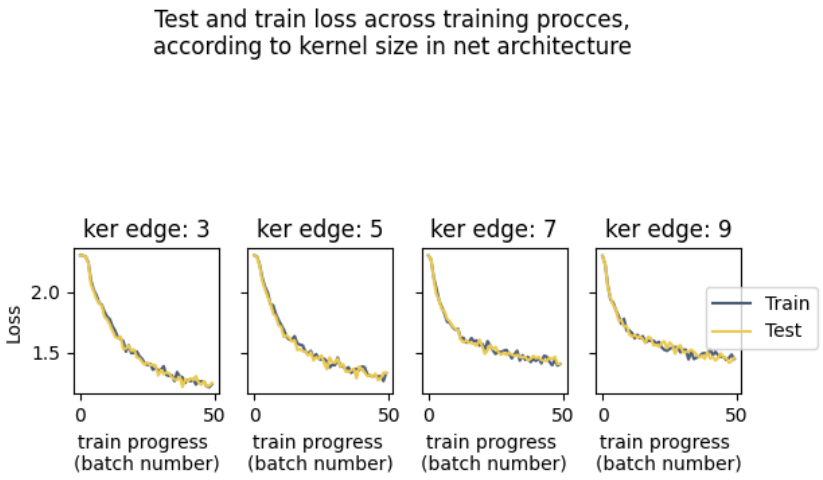
Ayala Prusak, Maya Cohen

March 2021

1 Architecture Fine-Tuning

For this section we trained nine different variations of the basic architecture. Four models were trained with the fixed 120 first FC layer and with kernel size $\in \{3, 5, 7, 9\}$ in both convolution layers. Five more were trained with the standard 5-edge sized kernels but with a large range on FC1 sizes $\in \{15, 30, 70, 100, 200\}$. The average loss in both the train and test sets was measured once in 500 batches, with batch size of 4 images.

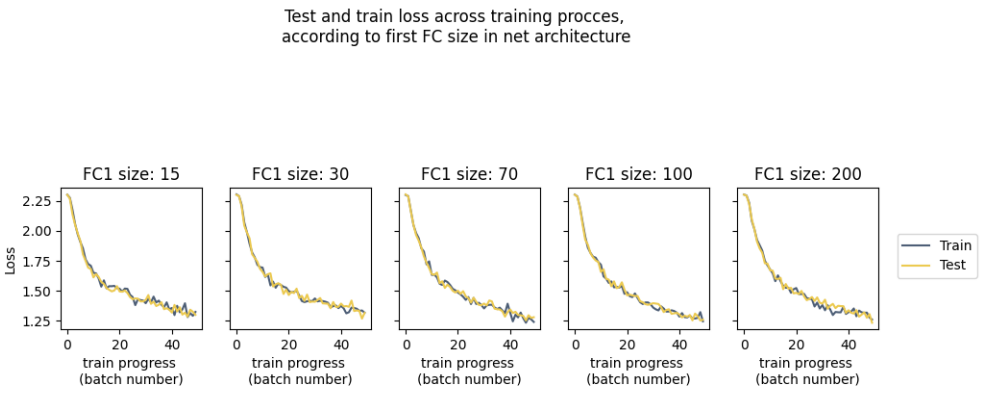
1.1 Optimal kernel size



From these result we conclude that the convolution layer doesn't tend to overfit, probably because of it's relatively small set of parameters and it's trans-

lation invariance that keeps it well balanced between different regions. In larger kernels we start to see signs for underfitting. We suggest that larger number of parameters is harder to learn, so as a result we observe a lower convergence rate. Therefore we decided to keep the 5-sized kernel as a satisfying architecture parameter.

1.2 Optimal FC size



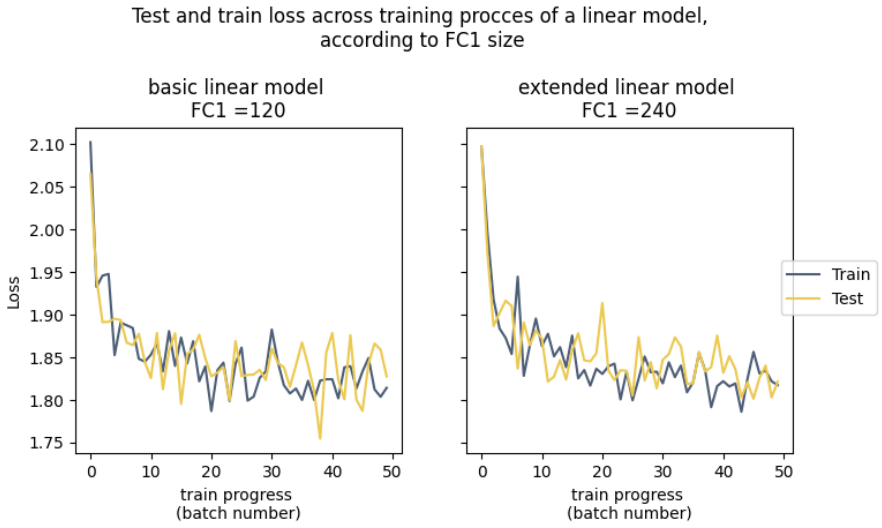
Similarly, while observing the range of FC1 sizes we see a minor underfit in sizes $15, 30$, and a slight overfit in the 200 experiment. We therefore assume that the optimal range for this parameter is within $70 - 100$.

2 Linear Model

2.1 Non linear operation

Next, we created a linear model. The linear model is similar to the basic architecture, omitting the non linear Relu and Max Pooling layers.

2.2 Results



It can be seen that this architecture outperforms previous versions. Since all of the non-linear layers where eliminated, we are left with a pure linear model that 'collapses' into a single linear operator. (Meaning it is equivalent to a weaker model, as adding more wights doesn't necessarily means higher expressiveness). Therefore this degenerated version has a limited 'learning capacity'. Extending this model by doubling it's FC1 layer didn't seemed to help at all.

3 Locality of the Receptive Field

In order to evaluate the importance of the receptive fields locality, it would not help to simply use larger filters. This is due to the fact that locality is an inherit trait of convolution operation, with the center of the kernel moving along the image.
So instead we randomly reshuffled the spatial locations of pixels as requested.
in order to asses the affect on the networks performance we would consider the following typical results:

3.1 Regular Net Performance [Without Shuffling]

(with the default parameters used in the tutorial)
Accuracy of the network on the 10000 test images: 54 %
Accuracy of plane : 52 %

Accuracy of car : 62 %
Accuracy of bird : 25 %
Accuracy of cat : 34 %
Accuracy of deer : 48 %
Accuracy of dog : 52 %
Accuracy of frog : 61 %
Accuracy of horse : 61 %
Accuracy of ship : 81 %
Accuracy of truck : 70 %

3.2 Net Performance With Shuffling

Accuracy of the network on the 10000 test images: 42 %
Accuracy of plane : 47 %
Accuracy of car : 57 %
Accuracy of bird : 21 %
Accuracy of cat : 15 %
Accuracy of deer : 32 %
Accuracy of dog : 52 %
Accuracy of frog : 47 %
Accuracy of horse : 38 %
Accuracy of ship : 68 %
Accuracy of truck : 43 %

The network would usually get about $\tilde{5}5\%$ accuracy when the pixels were in order and about $\tilde{4}0\%$ accuracy when the receptive field was de-localized. The accuracy was disparaged, since recognising reoccurring structures was made difficult. Yet still it is apparent the network is rather accurate even when disrupting the locality. We believe this has to do with the fact the images in the data base are relatively small, and their low resolution does not captor any local high frequency data. Also since we are always using the same permutation, local information in images of the same scale remains similar. when looking at the different classes we can notice that birds not very well detected but also that shuffling doesn't affect that, is is typical to most runs, leading us to assume there is not much local information used in bird recognition.

4 No Spatial Structure

In order to evaluate the networks performance with no special structure we have reshuffled each image every time if was loaded. This means the following results are only obtained from learning the color distributions of each class. as expected the network did not perform very well, but its results are still better then a guess, usually spanning around 23% accuracy.

4.1 Net Performance No Spatial Structure

Accuracy of the network on the 10000 test images: 26 %
Accuracy of plane : 44 %
Accuracy of car : 36 %
Accuracy of bird : 26 %
Accuracy of cat : 9 %
Accuracy of deer : 0 %
Accuracy of dog : 4 %
Accuracy of frog : 41 %
Accuracy of horse : 22 %
Accuracy of ship : 38 %
Accuracy of truck : 44 %

Looking at the different classes frogs are usually well detected even without spatial coherency, one could assum this has to do with their tendency to be green usually. This being said, the accuracy detecting frogs was very high to begin with.

5 Theoretical Questions

5.1 LTI is convolution

$$L[x(t)](y) = L\left[\sum_{\tau=-\infty}^{\infty} x(\tau)\delta(t-\tau)\right](y) = \sum_{\tau=-\infty}^{\infty} x(\tau)L\left[\delta(t-\tau)\right](y)$$
$$= \sum_{\tau=-\infty}^{\infty} x(\tau)L\left[\delta(t)\right](y-\tau) = \sum_{\tau=-\infty}^{\infty} x(\tau)h(y-\tau) = (h*x)(y)$$

In order to reveal the learned kernel parameters one may insert a signal from the following shape- a completely dark image (0-value pixels) with only a single 'alight' pixel with value 1 in the middle of it. The transformed image received by such input presents a mirrored version of the kernel parameters within a dark (empty) background. Notice the results is mirrors for both x and y axis.

$\star$ Due to inadequate prior knowledge in DSP we used an online resource for this question. (`dsp.stackexchange.com/questions/23988`)

5.2 FC input ordering

When reshaping a 2D activation map and feeding it into an FC layer, the order of the inputs no longer matters as long as it is consistent between train and test data. FC layers use independent weights for any input, so shuffling the inputs will eventually cause a different ordering of the learned weights, but not a meaningful change in the net's computation process.

5.3 LTI operators

5.3.1 ReLU

ReLU operator is translation invariant but not linear. For example:
$0 = ReLU(0) = ReLU(1 + (-1)) \neq ReLU(1) + ReLU(-1) = 1 + 0 = 1$

5.3.2 Strided pooling layer

Strided pooling layer is linear but not translation-invariant. Example: