NN for Images Ex1 - HUJI

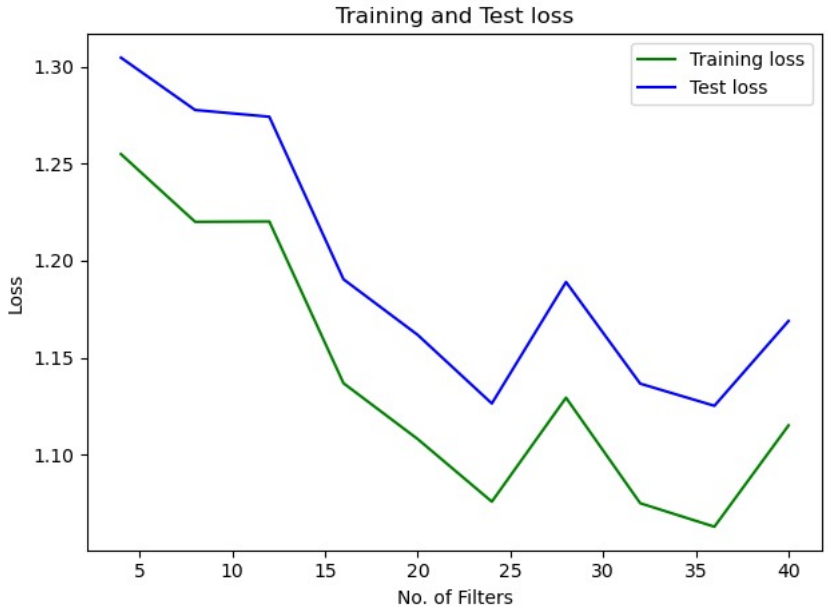
Alon Emanuel

April 12, 2021

1 Programming Task

1.1 Architecture Fine-Tuning

- To fine-tune our network architecture, we've ran the following procedure:
  - We trained 10 different networks, each with a **different number of filters**.
  - For each network, we saved the train and test losses, and plotted them as a function of the number of filters (see comment about notation below), to see which settings resulted in the best performance.
  - The change in the number of filters was done by **changing the depth of the first convolutional layer** (having it use less or more types of filters = having it output less or more channels).
- As can be seen in the plots, making the network too complex resulted in **overfitting**, while making it too simple resulted in **underfitting**.



- Note: the number of filters shown in the x-axis is notational only. The number of filters was in fact $n\_conv1 + n\_conv2 + n\_fc1 + n\_fc2$, where n_conv1 is the number of filters in the first convolutional layer.
  - This number is given by the formula $32 \cdot 32 \cdot out\_channels$. We fiddled with the $out\_channels$ parameter, and those are represented by the numbers that appear in the x-axis in the plot above.
- **With the** $conv1$ **output channels set to** 36**, we seem to get the best performance**, with the train loss being ≈ 0.9 and the test loss being ≈ 1.13.
  - The bump at 28 seems like an outlier and doesn't correspond to the expected bias-variance U-curve.

1.2 Linear Model

- After removing the non-linear components, the performance of the net **decreased drastically.**
- The components **include all the neuron activations** used in this architecture. More specifically, the ReLU activation was removed, and the original neuron emissions remained the same.
  - Note that as the *maxpool* layers are in fact non-linear, we chose to omit these from our test to put our focus on the main essence of the question (as opposed to removing the pooling layers and getting a completely different architecture).
- The results were as follows:
  - **Non-linear model** (with ReLU activations):
    - Train loss: 1.1437
    - Test loss: 1.2066
  - **Linear model** (without activation, and with 120 neurons in the first FC layer):
    - Train loss: 1.2391
    - Test loss: 1.2825
  - **"Deeper" linear model** (without activation, and with 240 neurons in the first FC layer):
    - Train loss: 1.2119
    - Test loss: 1.2571
- The big decrease in performance of the linear model is due to its **lack of expressiveness power.**
  - Since all layers in the linear model - convolutional, pooling and FC - are in fact linear operators, **they can all be chained together to create one linear operator** that'll perform the same operation.
  - That is, the number of filters and depth of the linear model doesn't have much effect after a certain point, since the expressive power can always be reduced to one single matrix multiplication.
- The non-linear components - the ReLU activations - **allow the model to express a completely new set of functions**, that can't be expressed-by or reduced to a simple matrix operation.

1.3 Locality of the Receptive Field

- Simply using larger filters won't help us test this aspect, since **larger filters are simply a generalization of smaller filters** - in the context of nerual networks.
  - This is due to the fact that while its training phase, the network can still update the filters' parameters to encode higher frequencies (smaller receptive fields).
  - This can be done by "centering" the filter's parameters and padding its edges.
  - By doing so, the network is not much different that a network with smaller kernels, thus the test is deemed unseccusful.
- When training the network on images with reshuffled pixels, we get a dramatic decrease in performance.
- When trained with the same network parameters as in the previous section, we get the following results:
  - **Original images**:
    - Train loss: 1.0736
    - Test loss: 1.13435
    - Test accuracy: 60.14
  - **Shuffled images**:
    - Train loss: 1.5758
    - Test loss: 1.6138
    - Test accuracy: 41.19

- I think this is the case since shuffled images deem the convolutional filters unusable. Or in other words - the underlying premise for the convolutional layers to work - is the locality of the receptive field.
- Conv. filters give a "score" to local area of the image (e.g. a 5x5 crop), based on how much that area/crop fits the filter, or doesn't fit.
- For instance, some conv. filters check for edges in a given crop of an image. Once these pixels are shuffled, there is no meaningful notion of "edge".
- Thus, **the network is "reduced" to its last FC layers**, who aren't affected by the shuffling of the images. This is due to its non-local nature. This is the reason we see results that are not purely random (a random net will have 10% accuracy).

1.4 No Spatial Structure

- Following the instructions, we've completely broke the image's structure, and compared the net's performance against the fixed-structured net:
  - Net trained on images with **fixed** permutations:
    - Train loss: 1.5348
    - Test loss: 1.5681
    - Test accuracy: 42.86
  - Net trained on images with **fresh** permutations:
    - Train loss: 1.9208
    - Test loss: 1.9142
    - Test accuracy: 27.21
- This result isn't surprising. The fixed permutations allowed the FC layers to "do their magic", but now when the permutations are freshly sampled with each image, even they can't work.
- We still get a better-than-random result, and my explanation will become clear once we look at the per-class results.
- **The per-class accuracies**, taken from the test set, are as follows:
  - Accuracy of plane : 55 %
  - Accuracy of car : 11 %
  - Accuracy of bird : 4 %
  - Accuracy of cat : 11 %
  - Accuracy of deer : 45 %
  - Accuracy of dog : 0 %
  - Accuracy of frog : 15 %
  - Accuracy of horse : 53 %
  - Accuracy of ship : 24 %
  - Accuracy of truck : 51 %
- As we can see, the plane, deer, horse and truck classes **did exceptionally well**, while bird, car, cat and dog failed miserably.
  - Images of planes, for example, share similar histograms - as the plane is usually white, and is photographed with a blue sky background.
  - The same goes for deer and horses (beige-brown over a green background) and probably for trucks, as they're usually white.
  - On the other hand - dogs, cats, birds and cars come in a variety of colors, and usually have changing backgrounds.
- This information about **consistent histograms is invariant to translations** and is obviously not local (but it should be noted that local histograms can be useful heuristics), and can be captured by the network, as weights get updated to reflect color distributions.

2 Theoretical Questions

Q1

- We need to provide some function f, such that

$$L\left[x\left(t\right)\right] \equiv \left(x * f\right)\left(t\right)$$

- First, recall the definition of a 1D convolution:

$$c\left[y\right] = x\left[y\right] * h\left[y\right] = \sum_{k=-\infty}^{\infty} x\left[k\right] \cdot h\left[y-k\right]$$

- First, lets decompose $x\left[t\right]$ into a weighted sum of translated delta functions:

$$x\left[t\right] = \sum_{k=-\infty}^{\infty} x\left[k\right] \cdot \delta\left(t-k\right)$$

- Now lets plug this into L:

$$\begin{aligned} L\left[x\left(t\right)\right]\left(y\right) =& L\left[\sum_{k=-\infty}^{\infty} x\left(k\right) \cdot \delta\left(t-k\right)\right]\left(y\right) \\ \overset{[1]}{=}& \sum_{k=-\infty}^{\infty} L\left[x\left(k\right) \cdot \delta\left(t-k\right)\right]\left(y\right) \\ \overset{[1]}{=}& \sum_{k=-\infty}^{\infty} x\left(k\right) \cdot L\left[\delta\left(t-k\right)\right]\left(y\right) \\ \overset{[2]}{=}& \sum_{k=-\infty}^{\infty} x\left(k\right) \cdot L\left[\delta\left(t\right)\right]\left(y-k\right) \end{aligned}$$

  - [1]: from L's linearity,
  - [2]: from the assumption over L.

- When assigning $f\left(t\right) \equiv L\left(\delta\left(t\right)\right)$,**we get the desired result**, showing that L corresponds to a convolution:

$$L\left[x\left(t\right)\right]\left(y\right) = \sum_{k=-\infty}^{\infty} x\left(k\right) \cdot f\left(y-k\right) = \left(x * f\right)\left(y\right)$$

Q2

- The order of the resulting 1D vector is unimportant.
- This is due to the fact that a FC takes into account **all connections between the previous layer** (the reshaped activation map) and the output layer.
- Thus, a neuron in the 2D activation map can be in any position, and the weights given to the edges going out of that neuron will change appropriately.

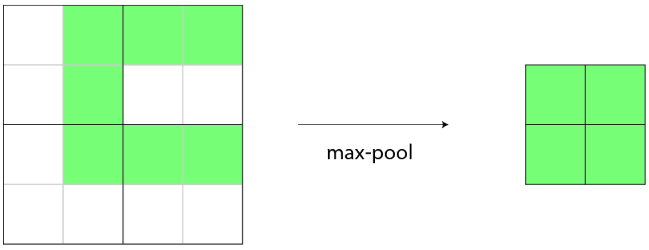
Q3.a

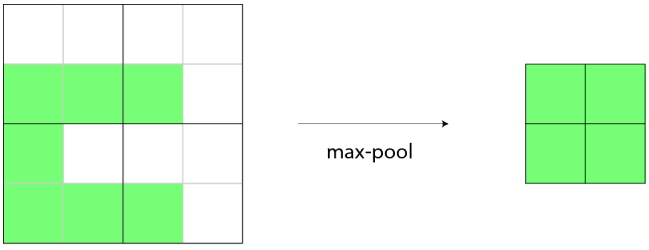
- The ReLU activation function **is not LTI**, and can be easily shown with the following example:
  - Given a neuron n_1 with emission set to 1, and given a ReLU activation layer, we get:

$$ReLU\left(\left(-1\right) \cdot n_1\right) = \text{ReLU}\left(\left(-1\right) \cdot 1\right) = 0 \neq -1 = \left(-1\right) \cdot \text{ReLU}\left(1\right) = \left(-1\right) \cdot \text{ReLU}\left(n_1\right)$$

Q3.b

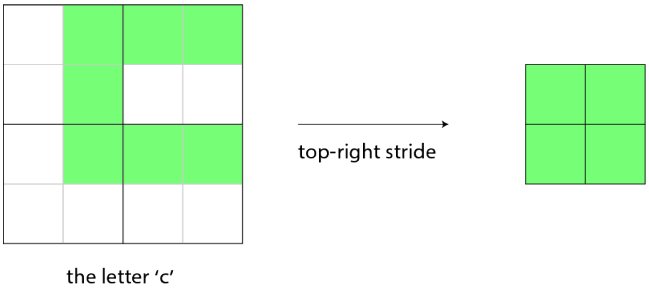
- The strided pooling layer **is not LTI**.
- To show this, we'll compare it to the translation-invariant (yet not necessarily linear) maxpooling layer:

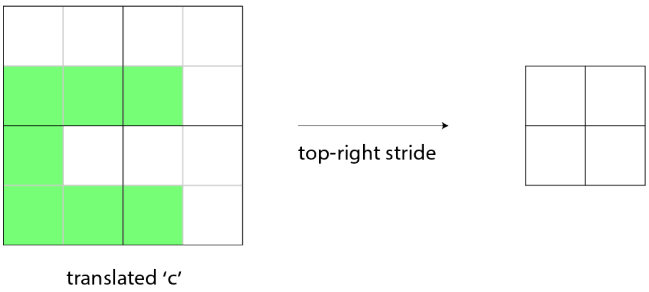




  - As can be seen, the maxpool layer kept the same features of the letter 'c', even after it was translated.

- In contrast, the strided pooling gave a completely different result, when the letter was moved by 2 pixels:





- Obviously this is a toy example, but it gets the idea across.