Neural Nearest Neighbour Replication

Sheng Zhong

April 23, 2019

Contents

1 Introduction

This report concerns the replication of Neural Nearest Neighbors Networks [1]. This work is worthwhile to replicate because it is the first continuous deterministic relaxation of performing k nearest neighbour (kNN) **selection**, where the outputs are the neighbours to query points. kNN selection is useful in many deep learning pipelines as a operation to pull in global information, but have so far been limited to either the start or end of the pipeline since gradients can't pass through the discrete action of selecting neighbours. By making this continuous and deterministic relaxation, the kNN selection becomes a differentiable operation that can be used in the middle of learning pipelines for applications such as image denoising and image correspondence.

My focus on replication is determining the core neural nearest neighbour's limitations and the effects of changing the relaxation level to kNN.

2 Work being replicated

The paper makes 3 contributions:

- Formulate a deterministic and differentiable relaxation of k nearest neighbour selection (kNN): neural nearest neighbours (NNN).
- Configure a sequence of network layers including NNN layers into a N^3 block, and further interleave N^3 blocks with other processing blocks such as DnCNN to create a N^3Net network architecture.
- Use the N^3Net architecture to perform image correspondence, denoising, and single image super resolution tasks and compare results against the state of the art.

2.a Neural Nearest Neighbours

kNN is a function mapping dataset X and a query set Y from the same space to a k-sized set from X for each query point in Y. In relaxing kNN to NNN, they also change the output to be "soft": we can define a meaningful gradient through it. NNN outputs k discrete probability distributions for each query point, with X being the domain of each distribution, representing the probability of that x value being the kth neighbour of y for some query point y and some input x of the kth distribution.

This output can be aggregated into the same form as the kNN output by taking the expected value of each distribution. This is not done in the intermediate stages of the N^3Net architecture.

The probability distributions are the output of taking a softmax [2] over the negative pairwise distance between points in X and Y as seen in Figure 1. They use Euclidean distance in all experiments; so do I.

They parameterize this relaxation by dividing the pairwise distances by **temperature** (t). This is a standard relaxation parameter in softmax, such that when $t \to 0$ softmax becomes argmax. Thus they make the argument that as $t \to 0$ NNN becomes kNN. We test

this claim in this replication. Temperature can be a fixed or learned parameter, and it can be either universal or data point specific.

$$\mathbb{P}\left[w^1 = i \mid \alpha^1, t\right] \equiv \text{Cat}(\alpha^1, t) = \frac{\exp\left(\alpha_i^1/t\right)}{\sum_{i' \in I} \exp\left(\alpha_{i'}^1/t\right)} \tag{3}$$

$$\text{where} \quad \alpha_i^1 \equiv -d(q, x_i). \tag{4}$$

Figure 1: Equations 3 and 4 from the paper for calculating the first probability distribution.

The other distributions are calculated similarly except for penalizing the locations where the previous distributions are active. This allows successive distributions (of the k distributions) to output different values.

2.b N^3 blocks and N^3Net

N^3 blocks are composed of sequences of alternating convolution, batch normalization, and relu layers. There are two independent branches of them inside each block, one for the input embedding, and the other for the temperature of each input feature.

The architecture specifics of a N^3 block can be found in Appendix A. The input to each N^3 block is a 8 layer feature from the output of a DnCNN block [3], and the output is a 64 layer feature. They use $k = 7$ for all NNN modules.

The N^3Net architecture is slightly different for each experiment. For denoising, it is made of a sequence of alternating N^3 blocks and DnCNN blocks. The first DnCNN block takes a 1 layer input (grayscale image), and the last DnCNN also outputs a 1 layer feature (denoised grayscale image). See tables 7-12 in the paper for more detail.

2.c Image experiments

In all experiments they operate on a greyscale image. They perform three experiments:

- Image denoising: an input image is corrupted by gaussian noise with a set sigma being the standard deviation of the noise. The goal is to recover the original image, evaluated with the metric of peak signal-to-noise ratio (PSNR) in decibels (dB). Producing high PSNR corresponds to removing more of the noise.
- Single image super-resolution (SISR): an input image is reduced in resolution through interpolation (bicubic in the paper) and the task is to recover the original image. The network is trained on other images such that it can "hallucinate" those details [4]. The same PSNR metric is used for evaluation.
- Image feature correspondence: classify correspondence between features on two images as correct or not. The mean average precision (MAP) metric is used for evaluation.

In all experiments, the paper finds significant improvements over the state of the art in the metric chosen.

3 Replication activities

Their code can be found https://github.com/visinf/n3net. It includes code for NNN, building a N^3Net , downloading the datasets, and running the image denoising experiment.

I reimplemented their NNN, but did not reimplement the infrastructure (N^3 and N^3Net) to use it; instead I swapped instantiations of their NNN with mine. This was because the infrastructure's complexity mainly comes from many technicalities, and I was more interested in the validity of their core method.

I trained two networks to perform image denoising: one using their parameters without modification, and the other with a normalization modification that I describe in 4.a.v. They were trained using the same random seed to test the effects of my modification. One hurdle was that training took around 10 minutes per epoch, and I needed to train for 51 epochs for each network to match the training in their paper. This was solved by training overnight.

The core of my work was investigating the limitations of their core NNN module and what effect temperature had on the approximation to kNN. I generated one dimensional datasets to test this because they were much more interpretable than images. After implementing my own NNN, I confirmed that it produced the exact same output as their NNN, so limitations I found were not an issue of my implementation. A hurdle here was conforming to their NNN interface, which was heavily engineered to support the N^3Net infrastructure and batch training. Thus the tensors passed around would always be 4 dimensional: $batch \times input \times output \times k$. The semantics of every tensor's input and output differed. This hurdle was solved by stepping execution through one run of applying their NNN.

4 Replication experiments and extensions

The extensions and experiments are one combined section because the results from the experiments naturally motivated the extension investigations which yielded further results.

4.a Deterministic Differentiable kNN

Temperature in experiments involving generated datsets is a universal non-learned value since it is often the manipulated variable.

4.a.i Functionality verification

We verify kNN-like functionality on simple data for ease of debugging and visualization. The data is an integer range with small additive random normal noise $X = \{0 + v_0, 1 + v_1, 2 + v_2, ..., N + v_N\}$. The noise is characterized by the standard deviation σ: $v_i \sim \mathcal{N}(0, \sigma^2)$. The reason for adding noise will become clear later in this section.

We consider the task of doing NNN with $k = 3$ querying with X itself (so $Y = X$). Since X is (nearly) integral, we expect each query to find the element itself and the 2 closest integers. Thus, the range inside each neighbourhood should be close to 2; we check this expectation over temperature values. From Figure 2 we see this convergence happening around $t = 10^{-5} = 0.00001$ for $\sigma = 0.0001, N = 10$.

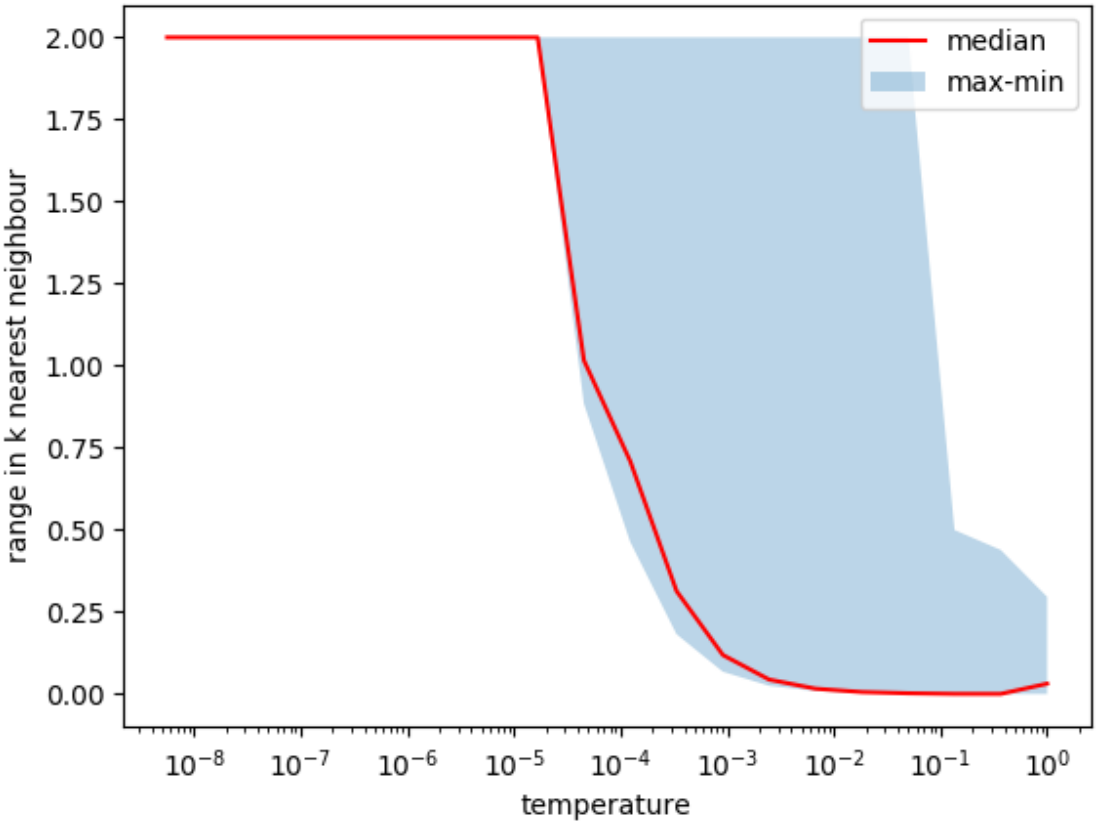

Figure 2: Range inside each neighbourhood relative to temperature with $\sigma = 0.0001, N = 10$. The blue shaded region shows the max and min values across all query points.

We also consider the average (across the N test points) minimum distance to a ground truth hard kNN implementation from `sklearn.neighbor`. This should converge to 0 as we converge to hard kNN, as verified in Figure 3. This happens at the same threshold as in 2 when we converged to the expected range.

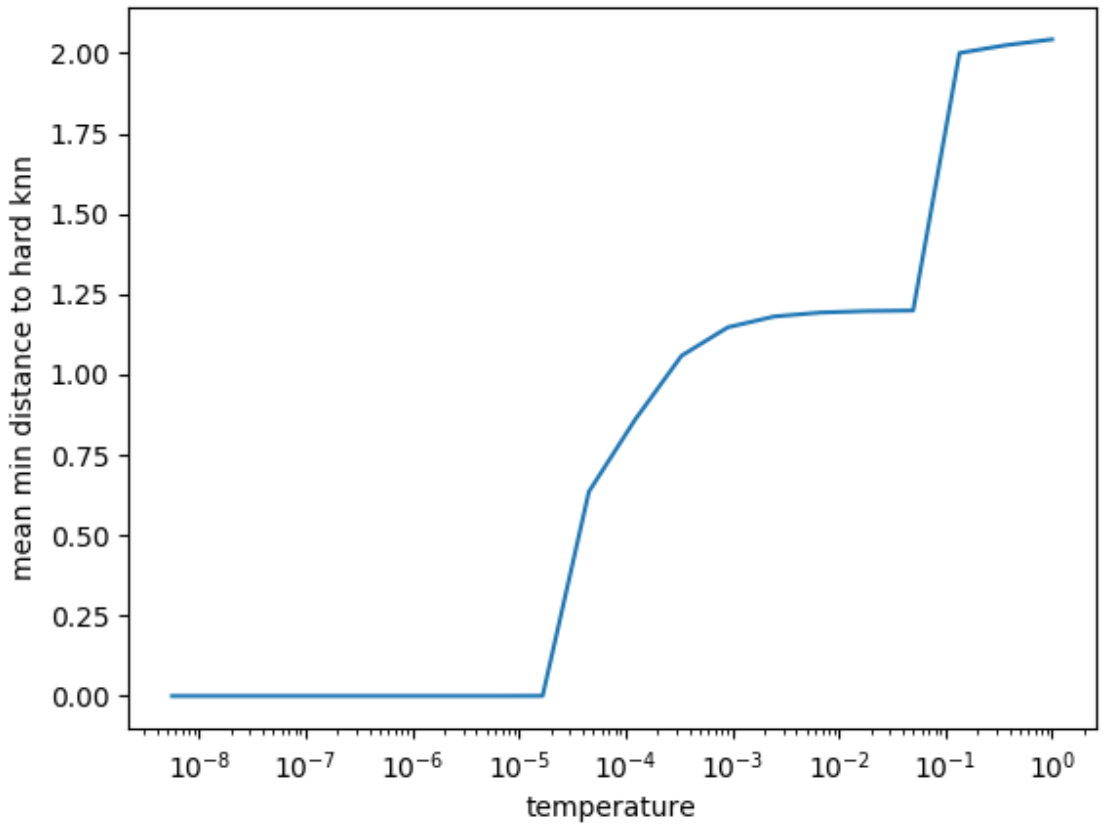

Figure 3: Average minimum distance from hard kNN in each neighbourhood with $\sigma = 0.0001, N = 10$.

4.a.ii Deeper convergence investigation

We investigate why the convergence happens so sharply around a threshold and how this threshold scales with other problem parameters.

A reminder that the soft outputs of NNN are the probability distributions of each data point being the kth neighbour. Each query point will produce k discrete distributions, each having X as the domain and summing to 1. We visualize how these distributions change with temperature. For simplicity, we consider only the k distributions (plotted in different colours) for a single query point ($y = 3$) shown in Figure 4.

The triangular peaks with lower temperatures are visualization artifacts since the distribution is actually discrete but the peaks remain even when almost all of the weight is on a single element. A histogram visualization overlapped too much across temperatures much to see changes. Each colour represents a different distribution corresponding to a different kth neighbour. We see that at high (≈ 1) temperatures the peak of each distribution is centered around 3, resulting in reporting the same element multiple times as different kth neighbours. This explains why the initial neighbourhood range is 0 in Figure 2.

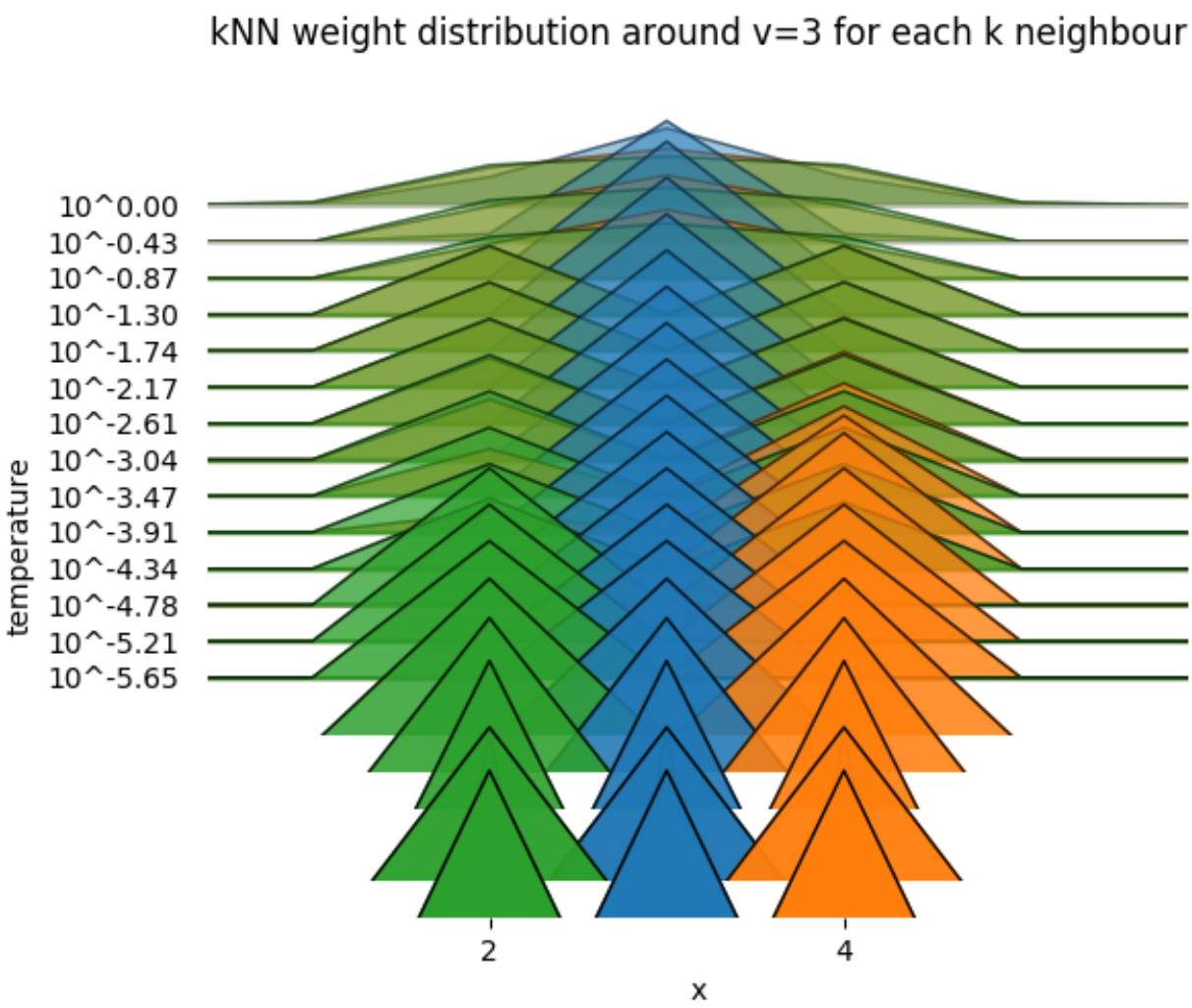

Figure 4: k neighbour distributions as we change temperature.

It's not until $t = 10^{-3.04}$ that we can visually distinguish the orange and green distributions, but the jump from $t = 10^{-4.34}$ to $t = 10^{-4.78}$ is dramatic as we go from the distributions being almost the same on both sides to being entirely on one side. I suspect this is related to the the weights being a softmax over logits of distance. Logit curves have low slope around 0.5 and grow to positive or negative infinity around 0 and 1, as illustrated in Figure 5. Since the distance of 2 and 4 to 3 is both 1, the difference in logit output is due to noise and takes a lot of amplification in terms of decreasing the temperature to distinguish these two equally valid neighbours.

We test this hypothesis by adjusting the standard deviation of noise σ in X while keeping the random seed the same to maintain noise direction (consider noise as a N dimensional vector). We measure the temperature threshold needed before one of the side distributions have a probability 10 times greater than the other in one of the neighbouring values, shown in Figure 6. As predicted, decreasing noise in X requires decreasing temperature to amplify the noise. The trend holds even when not keeping noise direction constant. Due to softmax exponentiating the logits and temperature, it is not too surprising the threshold relationship

looks affine in the log of both noise and temperature.

This experiment shows this relaxation of kNN is poor for data that have equidistant neighbours (data that have evenly spread out points).

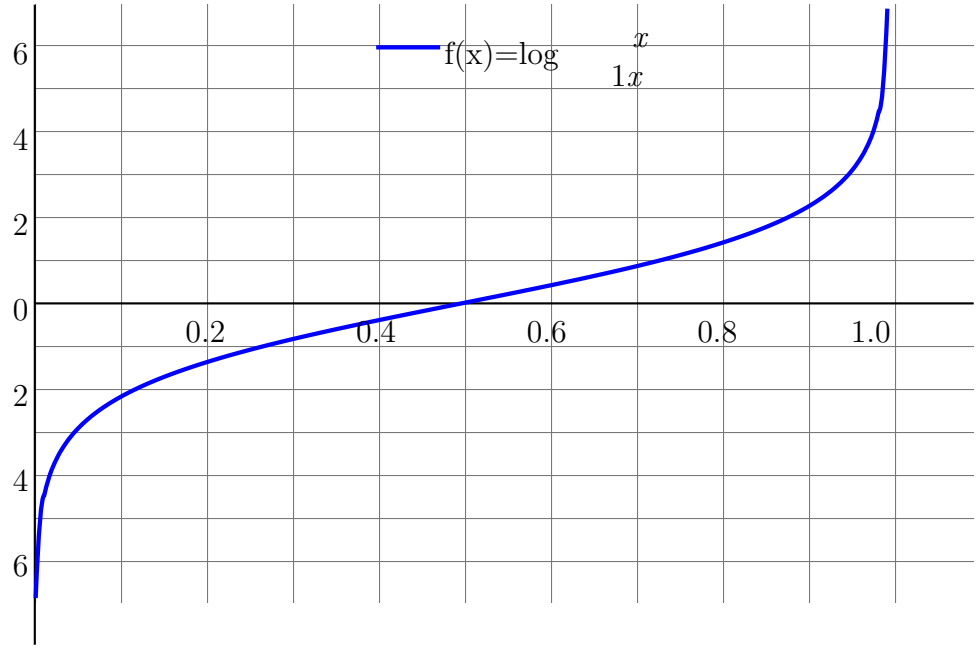

Figure 5: Logit function.

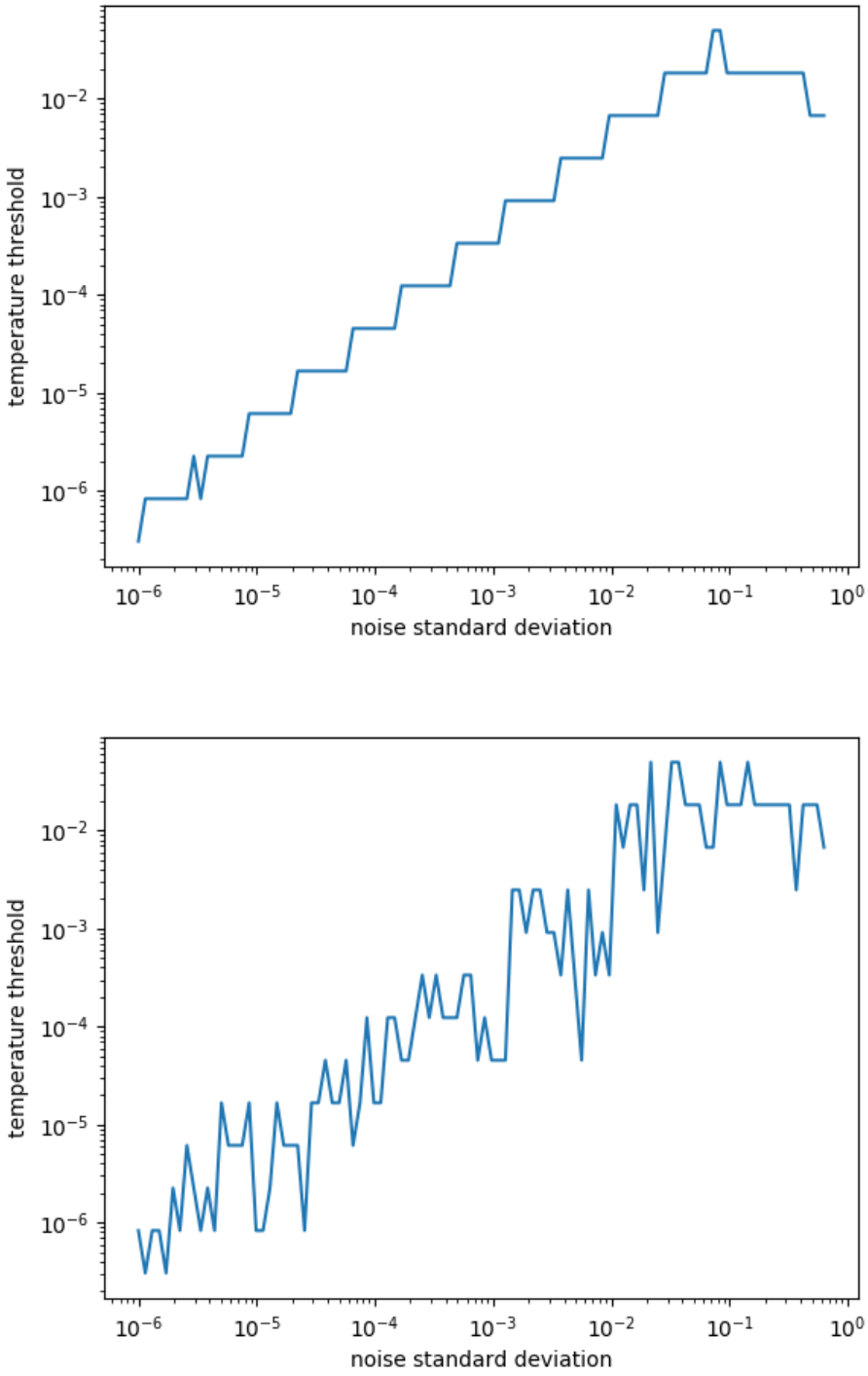

Figure 6: Temperature threshold for clear separation between NNN distributions relative to additive random noise standard deviation with (top) fixed random seed=123, and (bot) random random seed. Random seed corresponds to noise direction.

4.a.iii Runtime investigation

We consider runtime comparisons to hard kNN to evaluate if NNN is practical. The first experiment varies the dataset size N while keeping $k = 3$ and is shown in Figure 7. A major problem is that memory consumption quickly became intractable - $N \approx 10000$ needed $\approx$ 4GB of video memory (processed in CUDA). This is concerning for use on large datasets.

The paper proposes to consider only a limited patch around each query point for doing nearest neighbour on. This makes sense in some scenarios where we have an estimate of where the nearest neighbours could be, such as in images, but does not extend to all dimensions. We restrict the neighbour consideration to 6 within each x value. This allowed us to investigate up to $N = 10^4$, but beyond that we still run out of memory.

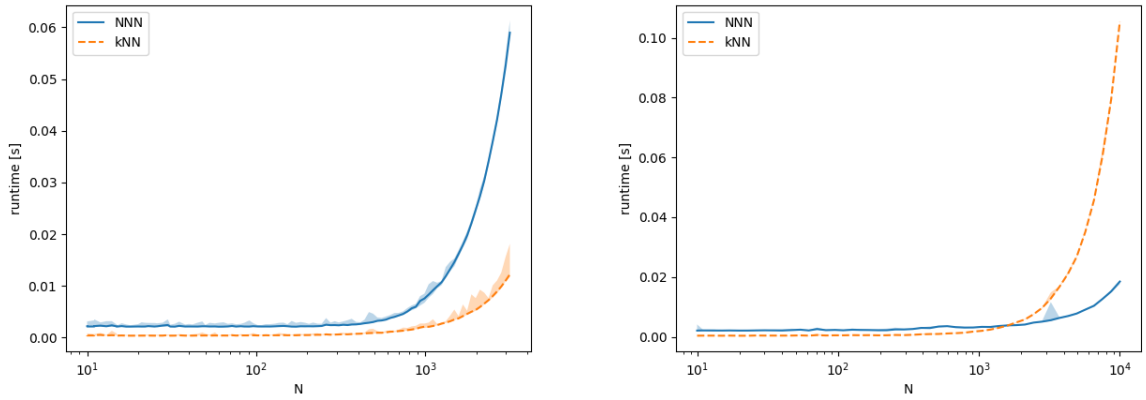

Figure 7: NNN runtime comparison against kNN. The shaded regions indicate the min and max over 20 trials while the lines represent the median. (left) NNN over all of X and (right) only over the 6 values around each query point (ordering known apriori).

4.a.iv Problem with decreasing temperature

As seen in Figure 1, temperature is the denominator of an exponential power for calculating the probability distributions:

$$P[w^{j+1}|\alpha^{j+1}, t] = exp(\alpha^{j+1}/t)/...$$

This has no theoretic problems, but runs into numeric implementation issues when $t \to 0$. This is demonstrated in Figure 8. As the figure shows, this problem is more significant with increasing N.

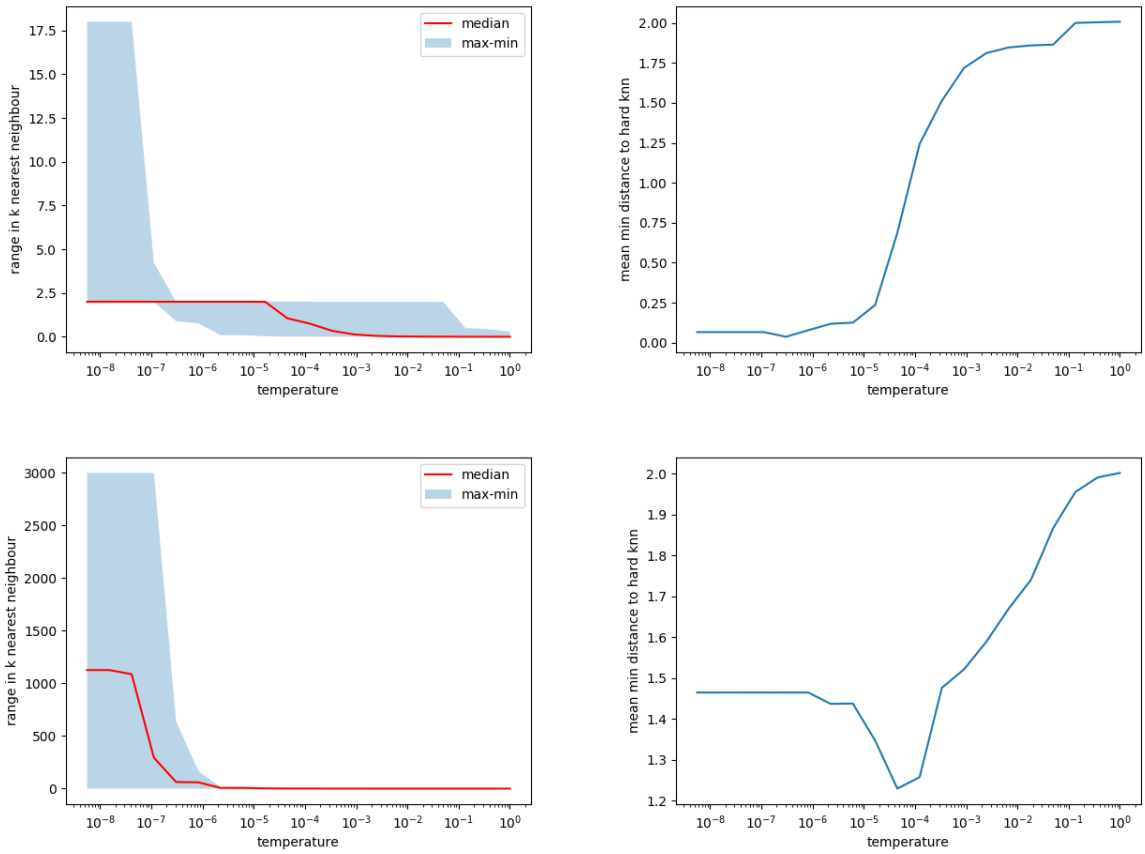

Figure 8: (left) Range inside each neighbourhood and (right) average minimum distance from hard kNN with $\sigma = 0.0001$, (top) $N = 30$, and (bot) $N = 3000$.

We end up with neighbourhoods that span pretty much the entire data range and do not converge to hard kNN. This is due to the appearance of spurious values from the softmax calculation resulting in the output not being proper distributions. For example, a specific case of this for value 22:

```
W[0,22]
tensor([...
        [0.0000, 0.3679, 0.3679],
        [1.0000, 0.0000, 0.0000],
        [0.0000, 0.3679, 0.3679],
        ...])
```

Each column is the output distribution for being the kth neighbour. The truncated rows were all 0. We see that only the first distribution is correct while the other ones need to be normalized.

4.a.v Distribution normalization

Normalization means making sure each distribution sums to 1; we do this by dividing each column by their sum, so that the above situation becomes:

```
W[0,22]
tensor([...
        [0.0000, 0.5000, 0.5000],
        [1.0000, 0.0000, 0.0000],
        [0.0000, 0.5000, 0.5000],
        ...])
```

Figure 9 shows the same experiment results with normalization.

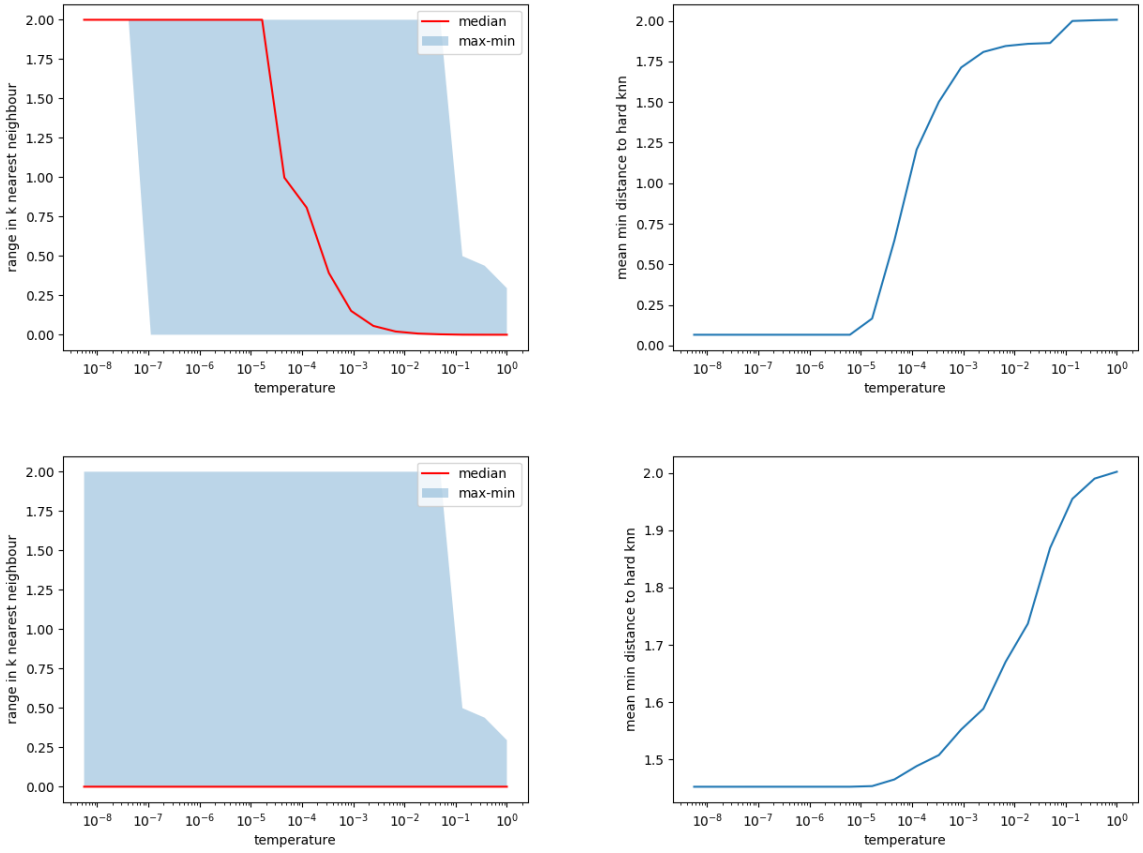

Figure 9: After normalization (left) range inside each neighbourhood and (right) average minimum distance from hard kNN with $\sigma = 0.0001$, (top) $N = 30$, and (bot) $N = 3000$.

While we've eliminated the problem of having spurious neighbourhood ranges, the performance in terms of being close to hard kNN did not improve. Since aggregating the distribution to produce a scalar output inherently **assumes a unimodal distribution**, it doesn't work when the distribution is bimodal as in this case since both neighbours are

equally valid. While this damages the usability of NNN as a soft replacement for kNN, the un-aggregated probability distribution output is still valid for use in the N^3 block.

4.a.vi Verification on randomly spaced data

While the previous section exposes NNN's weakness on evenly spaced data, we consider its performance on irregularly spaced data: $X_2 = \{v_0, v_1, v_2, ..., v_N\}$ σ: $v_i \sim \mathcal{N}(0, \sigma^2)$. From Figure 10 that we quickly converge to 0 error from hard kNN at around $t = 10^{-2}$.

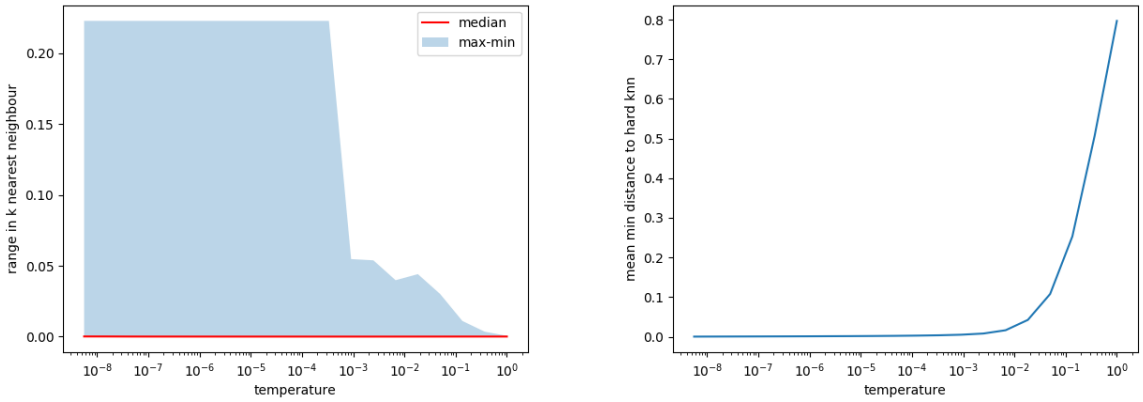

Figure 10: Randomly spaced data (left) range inside each neighbourhood and (right) average minimum distance from hard kNN with $\sigma = 0.5$, and $N = 3000$. In this case we have no expected value for the neighbourhood range.

4.b Denoising experiment

We train a network with the same parameters as in the paper and with a fixed random seed (1234) for 51 epochs (what the paper used). We compare PSNR with and without the normalization fix motivated by evenly spread out data. From Table 4.b, we see that our results are very similar to the published results (table 3 in the paper), and that normalization had no effect because the data was not evenly spread out and all the distributions were asymmetric.

Training is done on the entire BSDS500 dataset [5] augmented with random crops, random orientations, random vertical flips, and transformed to grayscale for a total of 1600 images. Figure 11 shows the training progress for one network.

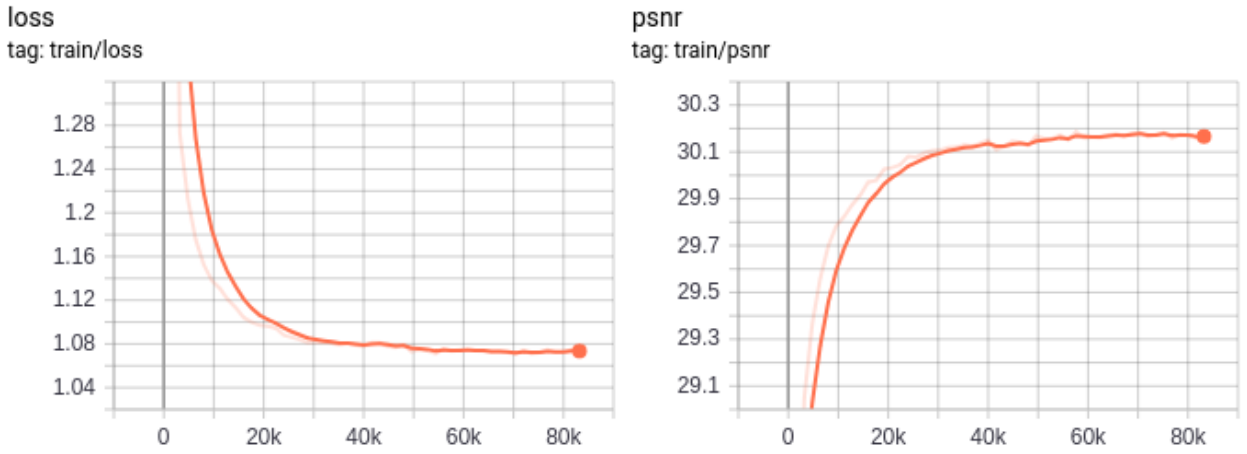

Figure 11: Loss and PSNR on the training set while training on 1600 augmented images from the BSDS500 dataset. The x axis corresponds to the number of images trained on.

Data	σ	Paper results	Replicated results	Replicated results with normalization
Set12	25	30.55	30.4997	30.4997
	50	27.43	27.4063	27.4063
	70	25.90	25.8790	25.8790
BSD68	25	29.30	29.2911	29.2911
	50	26.39	26.3912	26.3912
	70	25.14	25.1304	25.1304
Urban100	25	30.19	30.1783	30.1783
	50	26.82	26.8265	26.8265
	70	25.15	25.1513	25.1513

Table 1: Average PSNR across different datasets for the published result in the paper, our replicated result, and our replicated result with the normalization fix intended for symmetric distributions.

Evaluation takes about 1 hour on each network for a specific σ. The average runtimes for each image in each dataset are in 4.b.

Dataset	Replicated result	Replicated results with normalization
Set12	1.33	1.45
BSD68	1.63	1.32
Urban100	22.6	22.07

Table 2: Average runtime in seconds per image on each dataset. The trailing number of each dataset name is the number of images in that set.

5 Conclusion

Neural nearest neighbours is indeed a valid relaxation of k nearest neighbours controlled by the temperature parameter, approaching it as $t \to 0$. Allowing the temperature to be learned works well in image applications.

NNN breaks down in the case of evenly spaced data points, which is not fixable with the proposed distribution normalization. This is not a major obstacle for practical use since most real data are not evenly spread out. What is a major obstacle is the ridiculous memory usage scaling. It becomes intractable when the dataset is in the 10000 range (single dimensional). This limits NNN to applications where the input is constrained, such as on single images for denoising or super-resolution. This would not work on a growing data set.

An interesting future experiment is to test the effect of temperature on gradients through NNN. My hypothesis is that having $t \to 0$ also makes the gradients $\to 0$ since in the kNN limiting case we have no gradients. Intuitively this makes sense because the distributions being sharper with smaller t means apart from the single peak, the other parts of the domain will be very flat.

My original motivation for replicating NNN is to use in my own research pipeline. However the memory limitations are discouraging as my input set grows. Additionally, what I need is a soft kNN outputting a single distribution for each query point of which points would be one of the k nearest neighbours - I do not need to know the probability of a point being the specific kth neighbour. NNN gives that output and it comes at the cost of increased memory consumption and algorithmic complexity. During the replication I came up with my own kNN differentiable relaxation that does not give the unnecessary kth distribution information which I will be using in my own research.

A Appendix: N^3 Architecture

```
N3Block(
  (embedcnn): Sequential(
    (0): Conv2d(8, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.001, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.001, affine=True, track_running_stats=True)
    (5): ReLU(inplace)
    (6): Conv2d(64, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (tempcnn): Sequential(
    (0): Conv2d(8, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.001, affine=True, track_running_stats=True)
    (2): ReLU(inplace)
    (3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
    (4): BatchNorm2d(64, eps=1e-05, momentum=0.001, affine=True, track_running_stats=True)
    (5): ReLU(inplace)
    (6): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  )
  (n3aggregation): N3Aggregation2D(
    (aggregation): N3AggregationBase(
      (nnn): NeuralNearestNeighbors(
        (bn): BatchNorm1d(1, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
  )
)
```

References

[1] T. Plötz and S. Roth, "Neural nearest neighbors networks," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[2] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[3] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[4] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[5] B. S. D. Set, "Benchmarks 500 (bsds500)."