Neural Nearest Neighbour Network

Project ID: 15 | Robot Chitti

Abhigyan Gargav (2019102040)
Rohan Lahane (2019112004)
Raj Singh Parihar (2019102028)
Kartik Mehta (2019102029)

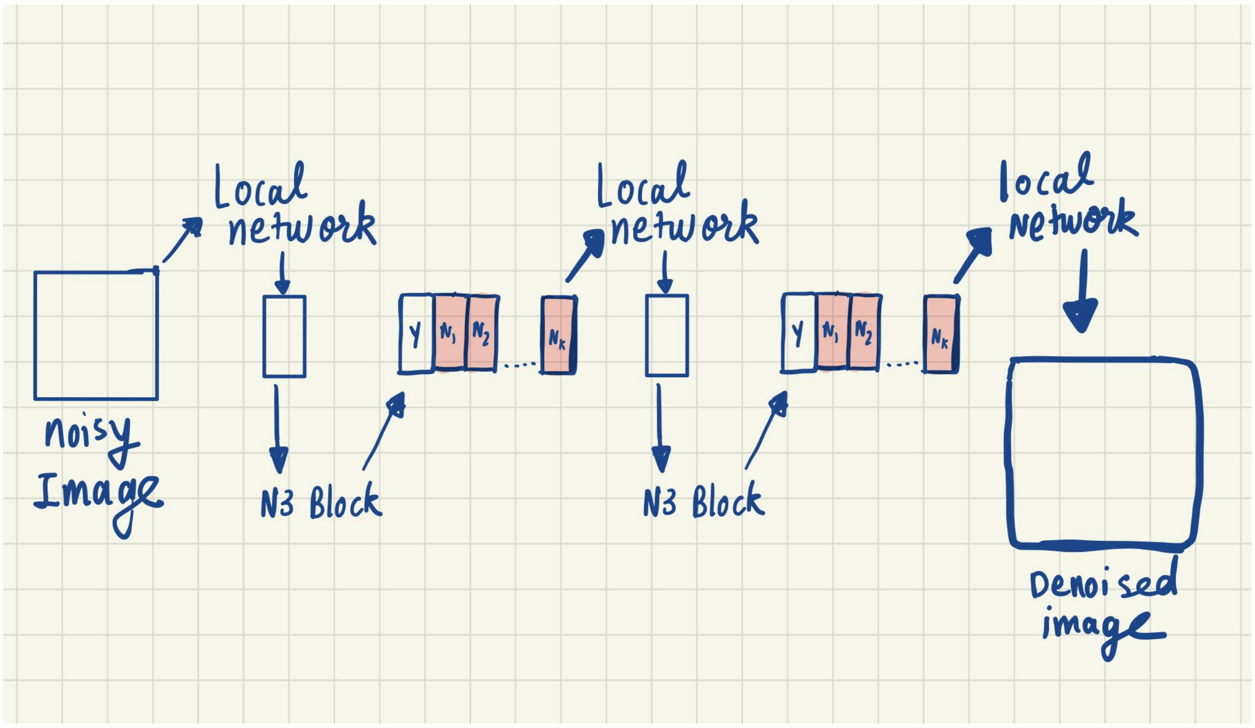
GitHub Link: https://github.com/kartik1365/SMAI-Project-Group

Prerequisites -

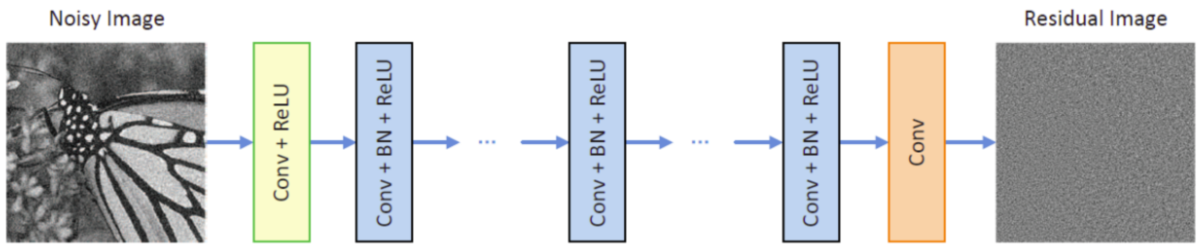
1. Dilated Convolution - Dilated Convolution is a technique that expands the kernel (input) by inserting holes between its consecutive elements. In simpler terms, it is the same as convolution but it involves pixel skipping, so as to cover a larger area of the input.
2. Receptive Field Size - It's defined as the size of the region in the input that produces the features. To obtain bigger receptive fields we generally use more convolution blocks which give more complex features in the feature space.
3. Self Similarity - Properties of image structures tend to reoccur within the same image, giving rise to a strong prior for image restoration.
4. Non-local Processing Methods like BM3D and non-local means to optimize the feature space. [Ref 3]
5. BM3D(Block Matching and 3D Filtering) - A non-local method for noise reduction.[Ref 4]

Neural Nearest Neighbour (N^3) Net

This is the main architecture that combines Local and Non-Local processing techniques to denoise a given image. It consists of these blocks in sequence, where the implementation of the individual blocks is described below.



Local Processing - DnCNN



DnCNN Network Architecture

DnCNN is a feed-forward denoising convolutional neural network to embrace the progress in very deep architecture, learning algorithm, and regularization method into

image denoising. Specifically, residual learning and batch normalization are utilized to speed up the training process as well as boost the denoising performance.
The size of convolutional filters are set to be 3×3 and all pooling layers are removed. Therefore, the receptive field of DnCNN with depth of d should be (2d+1)(2d+1).

Non Local Processing - Neural Nearest Neighbour (N^3) Block

Here we have implemented the neural nearest network block, in which First, an embedding network (a collective term used for mapping graph nodes of a vector to a multidimensional space) takes the output Y of the previous layer as input and calculates a pairwise distance matrix D between elements in Y as well as a temperature parameter(T).

KNN selection rule
Given a query item q, a database of candidate items $(x_i)\ i\ \in I$ with indices $I\ =\ \{1, 2, \cdots, M\}$for matching, and a distance metric $d(.,.)$ between pairs of items..
Where $\pi_q : I\ \rightarrow I$be a permutation that sorts the database items by increasing distance to q:

$$\pi_q(i_1)\ <\ \pi_q(i_2)$$

This implies, $d\ (q, x_{i_1})\ <\ d(q, x_{i_2})$

Where, $i_1, i_2\ \in I$

The KNN of q are then given by the set of the first $k$items w.r.t. the permutationπ_q

$$KNN(q)\ =\ \{x_i\,|\,\pi_q(i\ )\ \leq\ k\}$$

This rule is not differentiable. This does not allow us to derive gradients. So we will first show KNN as a limit of a parametric family of discrete stochastic sampling processes. Then we will derive continuous relaxations for the discrete variables. So we would eventually end up with a continuous deterministic relaxation.

KNN is interpreted as the limit of k categorical distributions. Let $Cat(w_1\,|\alpha_1, t)$be a categorical distribution, such that:

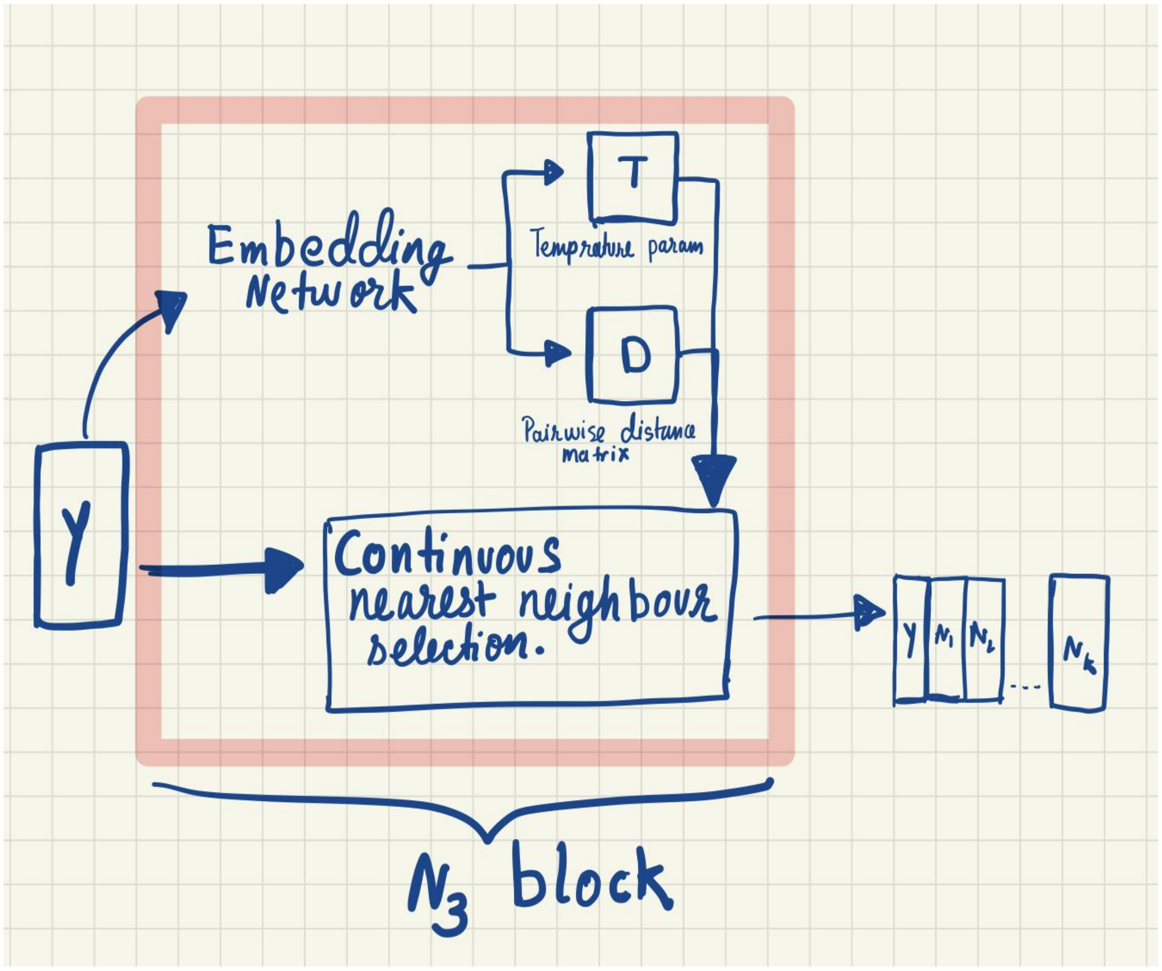
$$Cat(w_1\,|\alpha_1, t)\ =\ P[w_1 = i\,|\alpha_1, t]$$

Where, $\alpha_1\ =-\ d(q, x_i)$ and w= all zero vectors with 1 at the ith position.

In the limit $t \rightarrow 0$, $Cat(w_1 | \alpha_1, t)$ will converge to a deterministic distribution centred at the index of the database item with the smallest distance to q. Hence $Cat(w_1 | \alpha_1, t)$ is a stochastic relaxation of 1-NN. We need to generalise it for K-NN by constructing further distributions $Cat(w_{j+1} | \alpha_{j+1}, t)$.
Having derived KNN as a limit of a parametric family of discrete stochastic sampling processes, we will now derive continuous relaxations for the discrete variables.

Continuous deterministic relaxation
Replacing the weighted vectors w with their continuous expectations. This will give us a continuous relaxation of the stochastic nearest neighbours that still converges to the hard KNN selection rule in the limit case of $t \rightarrow 0$.
The expectation $\bar{w}_1$ of $w_1$is given by:

$$\bar{w}_1 = P[w_1 = i | \alpha_1, t]$$



Embedding network:

- Calculates (E = f_e(Y)) (feature embedding)
- Calculates distance matrix D such that D_{ij} = d(E_i,E_j)
- Another branch will calculate the temperature tensor (T= f_t(Y))

Continuous nearest neighbour selection:

- K continuous nearest neighbours can be estimated using D and T
- Ni and Y have the same dimensionality, so we can join them together

- To avoid reduction at this stage we will concatenate Y with k N_i

Note: Our implementation is not a trained model, thus it will give different results than in the paper due to differences in the random seed.

Classes and Functions used in code:

- `def cnn()` - This is the cnn processing required in the embedding network.
    - Variable ksize shows the size of the kernel.
    - The padding determines the variables excluded from the edges of the image
    - cnn_bn is the batch normalisation flag.
    - This emirates for the range of (cnn_depth -1) to extend cnn_layers from nn.Conv2d
    - If the depth of CNN is greater than 0 our function will append nn.conv2d to it,
    - Lastly, it will return the variable net which is just the CNN layers in sequential order.

- `Class N3Block()` - This is the implementation of the N3Block
    - It defines two things: Embedding network and Continuous KNN
    - It requires nn.Module as input.
    - First it defines the embedcnn function using cnn(), then finds the T tensor and D matrix. Secondly, it uses KNN (from non_local.py) to get the final output of N3Block.
    - The structure as given in image above is defined in forward() function.

- `Class DnCNN()` - This is the implementation of Local network using DnCNN architecture.
    - It is a standard and effective architecture

    - It mainly consists of the hourglass-like structure in the order from left to right as : *conv - relu - layers - conv.* This is defined in forward().

- `Class N3Net()` - This is the final class that binds all the substructures and blocks together to finally get the main net.
    - Takes *nblocks* as input, and defines *nblocks* numbers of DnCNN and N3Block in sequence as per the diagram given above.

- `non_local.py` consists of various classes and functions that the n3net needs for basic tasks like continuous K nearest neighbours implementation.
    - `def compute_distances` function computes the differentiable distance between two feature vectors (euclidean distance) which is further used to model the continuous KNN as a weighted average of features.
    - `class NeuralNearestNeighbours` consists of the implementation of the continuous and differentiable version of the KNN model as discussed in the paper.
    - This differentiability helps in integrating the KNN block in the CNN model without any difficulties during backpropagation.

References:-

1.Stamatios Lefkimmiatis. Universal denoising networks: A novel CNN-based network architecture for image denoising. In CVPR, pages 3204–3213, 2018.

2.Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In
CVPR, pages 60–65, 2005.

3.https://www.math.ucla.edu/~lvese/285j.1.09f/NonlocalMethods_main.slides.pdf

4.https://www.ipol.im/pub/art/2012/l-bm3d/article.pdf