Perturbative Neural Networks
(Supplementary Material)

Felix Juefei-Xu
Carnegie Mellon University
felixu@cmu.edu

Vishnu Naresh Boddeti
Michigan State University
vishnu@msu.edu

Marios Savvides
Carnegie Mellon University
msavvid@ri.cmu.edu

This article provides supplementary materials for [1]. In Section 1, we will show the proof of Sherman-Morrison formula. In Section 2, we will show the distance preserving property of PNN. In Section 3, computational and statistical complexity and efficiency of PNN will be discussed. In Section 4, we will discuss the relationship between noise level and convolutional filter size in detail. Finally, in Section 5, some visualization of intermediate responses of both PNN and CNN will be shown.

1. Proof of Sherman-Morrison Formula

The Sherman-Morrison formula [6, 7, 4] computes the inverse of the sum of an invertible matrix A and the outer product $uv^\top$ of vector u and v. Suppose $A \in \mathbb{R}^{n\times n}$ is an invertible square matrix and $u \in \mathbb{R}^n$, $v \in \mathbb{R}^n$ are column vectors. Then we have $A + uv^\top$ is invertible iff $1 + v^\top A^{-1}u \neq 0$. If $A + uv^\top$ is invertible, then its inverse is given by:

$$(A + uv^\top)^{-1} = A^{-1} - \frac{A^{-1}uv^\top A^{-1}}{1 + v^\top A^{-1}u} \tag{1}$$

If we already know the inverse of A, the Sherman-Morrison formula provides a numerically inexpensive way to compute the inverse of A corrected by $uv^\top$, which in our case, the correction can be seen as a perturbation in PNN. The computation is relatively inexpensive because the inverse of $A + uv^\top$ does not have to be computed from scratch, but can be computed by correcting or perturbing A^{-1}.

Proof:

($\Rightarrow$) To prove that the forward direction is true, we need to simply verify if the properties of the inverse hold: $XY = YX = I$, where $X = A + uv^\top$, and Y is the RHS of the Sherman-Morrison formula.

First, we verify whether $XY = I$ holds:

$$XY = (A + uv^\top)\left(A^{-1} - \frac{A^{-1}uv^\top A^{-1}}{1 + v^\top A^{-1}u}\right) \tag{2}$$

$$= AA^{-1} + uv^\top A^{-1} - \frac{AA^{-1}uv^\top A^{-1} + uv^\top A^{-1}uv^\top A^{-1}}{1 + v^\top A^{-1}u} \tag{3}$$

$$= I + uv^\top A^{-1} - \frac{uv^\top A^{-1} + uv^\top A^{-1}uv^\top A^{-1}}{1 + v^\top A^{-1}u} \tag{4}$$

$$= I + uv^\top A^{-1} - \frac{u(1 + v^\top A^{-1}u)v^\top A^{-1}}{1 + v^\top A^{-1}u} \tag{5}$$

$$= I + uv^\top A^{-1} - uv^\top A^{-1} \tag{6}$$

$$= I \tag{7}$$

In the same way, we can verify the following:

$$YX = \left(A^{-1} - \frac{A^{-1}uv^\top A^{-1}}{1 + v^\top A^{-1}u}\right)(A + uv^\top) = I \tag{8}$$

($\Leftarrow$) To prove the backward direction, we suppose that $u \neq 0$, otherwise the result is trivial. Then we can have the following identity:

$$(A + uv^\top)A^{-1}u = u + uv^\top A^{-1}u = (1 + v^\top A^{-1}u)u \tag{9}$$

Since $A + uv^\top$ is assumed to be invertible, it implies that $(A + uv^\top)A^{-1}$ is also invertible (as the product of full rank matrices). So, by our assumption that $u \neq 0$, we have that $(A + uv^\top)A^{-1}u \neq 0$, and by the identity shown above, this means that $(1 + v^\top A^{-1}u)u \neq 0$, and therefore $1 + v^\top A^{-1}u \neq 0$, as was intended to be shown.

2. Distance Preserving Property of PNN

PNN also has **distance preserving property**. For input $\mathbf{x}_p$ and $\mathbf{x}_q$, the PNN produces outputs $\hat{\mathbf{y}}_p$ and $\hat{\mathbf{y}}_q$ as follows:

$$\hat{\mathbf{y}}_p = \sum_{i=1}^{N}(\mathbf{x}_q + \mathbf{n}_i)v_i = (\mathbf{x}_p\mathbf{1}^\top + \mathbf{N})\mathbf{v} = \mathbf{x}_p\sum_{i}^{N} v_i + \mathbf{N}\mathbf{v}$$
$$\hat{\mathbf{y}}_q = \sum_{i=1}^{N}(\mathbf{x}_q + \mathbf{n}_i)v_i = (\mathbf{x}_p\mathbf{1}^\top + \mathbf{N})\mathbf{v} = \mathbf{x}_q\sum_{i}^{N} v_i + \mathbf{N}\mathbf{v}$$

we now examine the distance between $\hat{\mathbf{y}}_p$ and $\hat{\mathbf{y}}_q$ as:

$$\begin{aligned}\|\hat{\mathbf{y}}_p - \hat{\mathbf{y}}_q\|_2^2 &= \left\|\mathbf{x}_p\sum_{i}^{N} v_i + \mathbf{N}\mathbf{v} - \mathbf{x}_q\sum_{i}^{N} v_i - \mathbf{N}\mathbf{v}\right\|_2^2 \\ &= \left(\sum_{i}^{N} v_i\right)^2 \cdot \|\mathbf{x}_p - \mathbf{x}_q\|_2^2\end{aligned} \tag{10}$$

Applying Cauchy-Schwarz inequality, we can have:

$$\|\hat{\mathbf{y}}_p - \hat{\mathbf{y}}_q\|_2^2 \leq N \cdot \|\mathbf{v}\|_2^2 \cdot \|\mathbf{x}_p - \mathbf{x}_q\|_2^2 \tag{11}$$

3. Computational and Statistical Complexity

In comparison to standard CNNs, PNNs can be very attractive from the perspective of computational and statistical efficiency.

3.1. Computational Efficiency

Computationally, PNNs are significantly more efficient than CNNs, since PNNs completely do away with expensive convolution operations and instead replace them by more efficient weighted linear combinations. Given an input $k \times m \times n$, a convolutional layer with weights $k \times 3 \times 3$ and a corresponding PNN layer with weights $k \times 1 \times 1$, the number of multiplications required by the PNN layer is $= kmn$, while for the CNN needs $\approx 3kmn$ multiplications when using Winograd convolution [2, 8]. In practice, however, we did not observe any speed-up. We believe that this may be since 3×3 convolutions enjoy highly optimized implementations, while the 1×1 convolution implementations are not explicitly optimized and typically suffer from cache misses. Storage wise PNN result in smaller models, due to the fewer number of parameters (by a factor of pq) in a PNN layer in comparison to a CNN layer with the same number of input and output channels.

3.2. Computational Complexity

The weighted linear combination operation of PNN can be implemented either through a 1×1 convolution with the NCHW data format, or through a fully connected layer with the NHWC data format, with the latter being the more efficient implementation (up to a factor of $2\times$ in our experiments). However in a full network like ResNet, we are constrained to implementing PNN as a 1×1 convolution since the other layers in ResNet are optimized (NVIDIA's cuDNN) only for the NCHW data format. Furthermore, We stress that current cuDNN implementation of conv2d is highly optimized for 3×3 convolutions with the NCHW data format. We believe that optimized implementations will allow us to realize the full computational benefit of PNN over 3×3 convolutions. Our current implementation does the following, (1) transforms the

NCHW data into NHWC format, (2) performs a matrix-matrix multiplication (fully connected layer) mapping p-channels to q-channels, and (3) transforming the data from NHWC back to NCHW. Note that while the fully connected layer is faster than the 1×1 convolution, transforming the data between the NCHW and NHWC formats is an expensive operation. This new implementation demonstrates a modest 10% speed-up over standard 3×3 convolution.

3.3. Statistical Efficiency

The lower model complexity of PNN makes them an attractive option for learning with low sample complexity. To demonstrate the statistical efficiency of PNN, we perform additional face recognition tasks on the FRGC v2.0 dataset [3] under a limited sample complexity setting. The total number of images in each class ranges from 6 to 132 (51.6 on average). While there are 466 classes in total, we experiment with increasing number of randomly selected classes (10, 50 and 100) with a 60-40, 40-60, and 20-80 train/test split. Across the number of classes and various train/test splits, our network parameters remain the same except for the classification fully connected layer at the end. For both the PNN and CNN we adopt the ResNet-18 architecture with the same number of input and output channels. While the CNN architecture uses 3×3 filters, our PNN network uses 1×1 filters, therefore the PNN consists of much fewer learnable parameters ($9\times$ fewer) compared to the CNN. We make a few observations from our experimental results (see Figure 1): (1) PNN converges faster than CNN. This is true across all number of classes as well as all train/test splits. (2) PNN outperforms CNN on this task. Lower model complexity helps PNN prevent over-fitting especially on small to medium-sized datasets.

4. Noise Level and Convolutional Filter Size

As we have shown in Section 3.4 of the main paper [1], the difference (ϵ_i) between neighboring pixels can be treated as a random variable with $E(\epsilon_i) = 0$, $E(\epsilon_i^2) = 2\sigma^2(1-\rho) = 2\sigma^2\delta$, and $E(\epsilon_i\epsilon_j) = \sigma^2(1-\rho) = \sigma^2\delta$. Therefore we can further derive the correlation coefficient ρ_ϵ between ϵ_i and ϵ_j as follows:

$$\rho_\epsilon = \frac{E(\epsilon_i\epsilon_j)}{\sqrt{E(\epsilon_i^2)}\sqrt{E(\epsilon_j^2)}} = \frac{\sigma^2\delta}{\sqrt{2\sigma^2\delta}\sqrt{2\sigma^2\delta}} = \frac{\sigma^2\delta}{2\sigma^2\delta} = \frac{1}{2} \tag{12}$$

The derivation in Section 3.4 suggests that the choice of distribution of the noise ϵ_i in the PNN formulation can potentially be flexible, as long as its first and second order statistics behave according to the expressions shown in the main paper [1].

Next, we will discuss how the additive perturbation noise level (variance of the noise) in PNN can be related to the size of the convolutional filter in traditional CNN. More specifically, adding noise with low variance is equivalent to using a small-sized convolutional filter, while adding noise with larger variance is equivalent to convolving with a larger filter. Therefore, by adjusting the noise distribution (variance) PNN can potentially mimic a CNN with different filter sizes using different noise variance in each channel of the perturbation noise layer without having to explicitly decide on the filter sizes throughout the network layers.

Let us revisit the equivalent noise, $\sum_{i\in\mathbb{N}_c} \epsilon_i w_i$, in a CNN formulation in Eq. 24 of the main paper [1], or equivalently $\sum_{i\in\mathbb{N}_c} \epsilon_i w_i'$ with a normalization term divided on both sides of the equation. It can be seen that in Eq. 24 as a larger-sized convolutional filter is applied, more $\epsilon_i w_i$ terms are involved in the equivalent noise expression.

Let us assume that random variables ϵ_i's are Gaussian and according to the aforementioned discussion, they are not independent, with correlation coefficient ρ_ϵ. The sum of two non i.i.d. Gaussian variables $\epsilon_i + \epsilon_j \sim \mathcal{N}(0, E(\epsilon_i^2) + E(\epsilon_j^2) + 2E(\epsilon_i\epsilon_j)) \sim \mathcal{N}(0, 2\sigma^2\delta + 2\sigma^2\delta + 2\sigma^2\delta)$. An extension to this is that the linear combination of two non i.i.d. Gaussian variables $w_i\epsilon_i + w_j\epsilon_j \sim \mathcal{N}(0, w_i^2E(\epsilon_i^2) + w_j^2E(\epsilon_j^2) + 2w_iw_jE(\epsilon_i\epsilon_j)) \sim \mathcal{N}(0, w_i^2 2\sigma^2\delta + w_j^2 2\sigma^2\delta + 2w_jw_j\sigma^2\delta)$. Therefore, adding up more terms as the convolutional filter size becomes larger would result in a Gaussian distributed perturbation noise with larger variance.

The above analysis is also verified empirically using random convolutional filters of various sizes. We have randomly collected 500 images from randomly chosen 100 classes from ImageNet [5]. We have also generated 100 uniformly distributed random convolutional filters of sizes 3×3, 5×5, and 7×7 each. Images and convolutional filters are single-channeled for simplicity. We have convolved the 500 images with each set of 100 convolutional filters and study the statistics of the equivalent perturbation noise maps. The statistics are shown in Table 1 and the histograms are shown in Figure 2. As can be seen, empirically, larger filter size does correspond to larger noise level being added.

5. Visualizing Intermediate Responses

Finally, we provide a visual comparison (see Figure 3) of the intermediate feature maps of CNN and PNN for a ResNet-18 architecture trained on the ImageNet dataset. We observe that in the initial layers, the feature maps of the CNN and PNN both

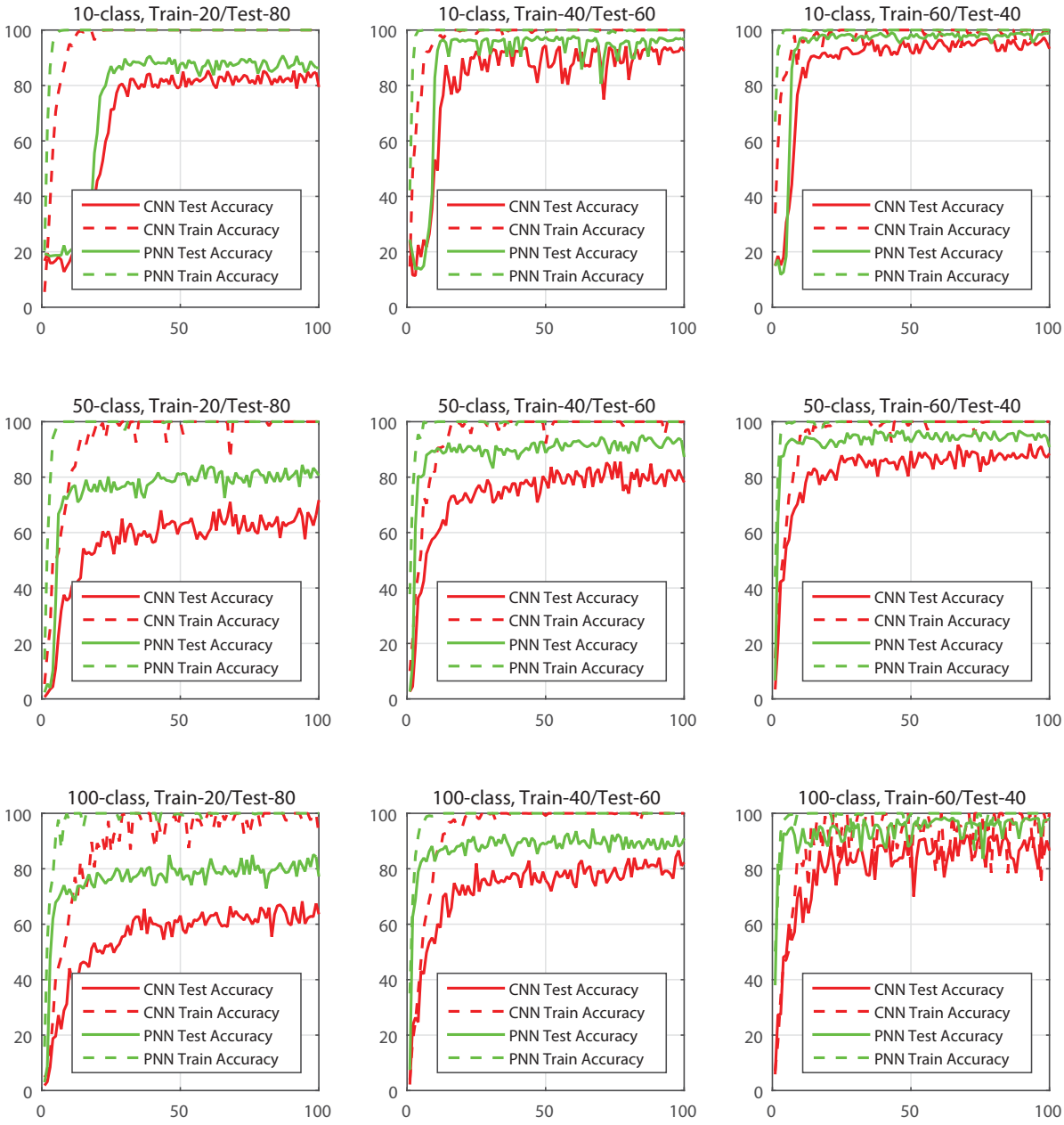

Figure 1: Classification on FRGC dataset with low sample complexity. Rows 1-3: 10-class, 50-class, and 100-class classification problems. Columns 1-3: 20-80, 40-60, and 60-40 train/test splits. In general, PNN shows statistical efficiency over CNN.

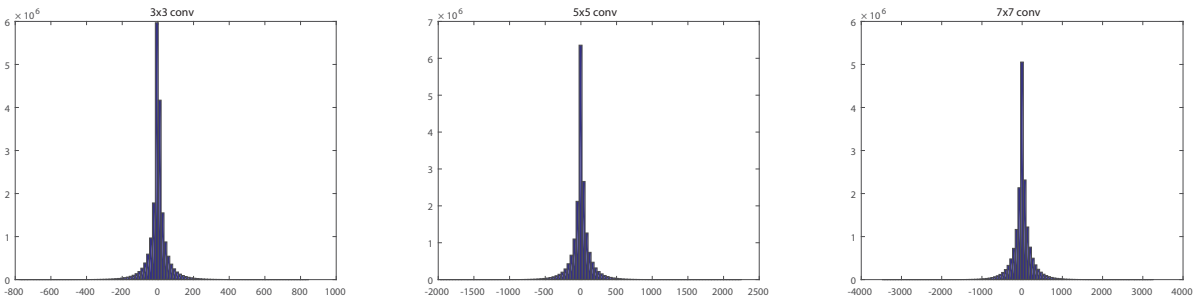

Figure 2: Histograms of the difference maps for various convolutional filter sizes. Please note the different x-axis limits.