Unsupervised Deep Learning Solutions For Computer Vision Tasks: Noise2Patch

---

Jacob Zimmerman

April 28, 2022
Version: Draft Version Name
Advisor: Jacob Zimmerman

Abstract

Discriminative deep learning models have dominated the world of computer vision for a generation. Neural networks serve as a basis for tasks like denoising, super-resolution, inpainting, image-dehazing, watermark-removal, and much more. However, these neural networks require great deals of data to work. Acquiring images can often be tedious, expensive, or even impossible. For example, the limitation of clean data is virtually ubiquitous in biomedical tasks, where a plethora of noisy data is available, but clean data is scarce. Thus, the need for unsupervised learning techniques arises when only noisy data is available or intensive data collection is too costly. A method named Noise2Noise found that neural networks can actually use noisy images as the input *and* the target because the expected value of a noisy image is the ground truth. In other words, the noisy targets will represent the ground truth on average. Another paper, Noise2Void (N2V), leverages this discovery by creating a self-supervised method using patches of pixels from an image that have a single, central pixel censored. The patch with a pixel censored is used as the input and the censored pixel is used as a target. N2V is among the highest performing unsupervised learning methods for denoising. Here, we introduce a method very similar to N2V: Noise2Patch (N2P). Instead of predicting one pixel from one patch, N2P predicts one entire censored patch from many random patches from the input image during training. Each random patch has an associated weight based on its color and distance from the censored patch. A color and position more similar to the censored patch results in a higher weight. ADD SOMETHING ABOUT HOW IT PERFORMED HERE

Contents

1

Introduction

1.1 Motivation

Unsupervised Deep Learning for Computer Vision such as ours find creative ways to leverage neural networks (typically a U-Net) to accomplish computer vision tasks like denoising, inpainting, super-resolution, and more (although those three are the most researched) without any clean data and varying amounts of "dirty" ("dirty" means noisy in the case of denoising, for example). Some methods, such as Deep Image Prior [Ulyanov et al. 2017], utilize no data at all, and only train a neural network on one dirty image. Our method aims to do the same. The most prominent application of such methods is for biomedical image data, where little to no data is available due to volatility in the living biological specimens. However, there can be many applications, especially for people with limited resources to allocate towards collecting data. Taking a more economic perspective, these methods give smaller institutions a chance to compete with large corporations who have a long history of data collection and more resources.

1.2 Background in Neural Networks.

The modern neural nets geared toward computer vision tasks in deep supervised learning typically use pairs of dirty-clean RGB images, where dirty and clean are task dependent terms (for example, dirty means noisy and clean means without noise for denoising), (x^i, y^i), to train their models (x^i is the same image as y_i with some sort of added "dirt", such as noise, that we are trying to remove). The idea is to minimize the loss, or unwanted difference, between our prediction of the clean image $f_\theta(x^i)$

(output of our model with parameters θ), and the true y_i. This is achieved by finding the model parameters θ^*, such that:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_i \mathcal{L}(f_\theta(x^i), y^i), \tag{1.1}$$

where the loss $\mathcal{L}$ is typically a norm used to measure the difference between $f_\theta(x^i)$ and y^i. Common norm choices are L_2 (Euclidean Distance, often denoted by $\|\cdot\|$), L_1 (Manhattan Distance), and L_0 (Quan et al. 2020). One possible way to solve this equation is using stochastic or batch gradient descent:

1. Calculate the gradient of the loss $\nabla\mathcal{L}(f_{\theta_t}(x), y)$ with respect to each of our parameters θ_t at time step t after plugging in a data point or a batch of data points.

2. Update each parameter in θ_t of our model with this equation:

$$\theta_{t+1} = \theta_t - \eta\nabla\mathcal{L}(f_{\theta_t}(x), y) \tag{1.2}$$

   where η is a constant called learning rate. Often, chosen ahead of time, the learning rate determines how large the step size is. A learning rate/step size too large will cause our parameters to continuously jump around to different values and be unable to converge, but too small may mean that the parameters have converged to a local minimum.

3. Repeat until θ converges to θ^*.

1.2.1 Perceptrons

A perceptron is a single layer neural network. In computer vision, the input is usually pixels. In binary classification a perceptron, each pixel has a value (color) and an associated weight that makes up the model parameters θ, which will be optimized during training. The summation of all value-weight products is then run through an activation function. An activation function has a binary output, either 1 or 0, that classifies the image based on whether the input value reaches a certain threshold.

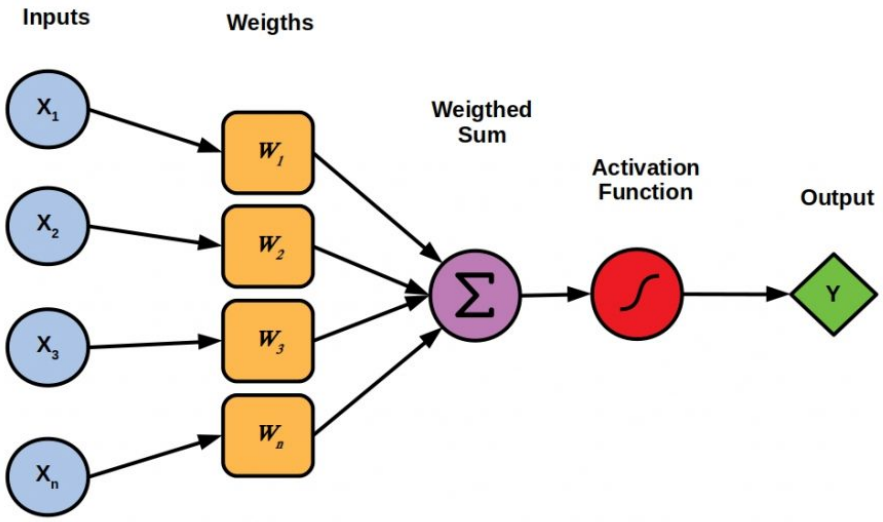


Fig. 1.1: An example of a perceptron. Image from RainerGewalt and * 2021.

Classifying greyscale images as either dog or cat is one use case of a perceptron. Say we had a set of greyscale images of either a cat or a dog with pixel values of 0 to 255 as well as each image's corresponding true label. During training, each training image is passed through the network and the summation is calculated. Say the summation came out to .7 and the answer was dog = 1. The loss would be .3. If the answer was cat = 0, the loss would be -.7. The gradient of the loss with respect to the model parameters θ is calculated for use in gradient descent and optimization of θ for the image-label training data. After convergence of θ, we have a model that outputs either dog or cat for a given grey scale image.

1.2.2 Multi-Layer Perceptron

Single-layer perceptrons effectively create a linear decision boundary from the training data, as shown in Figure 1.2.

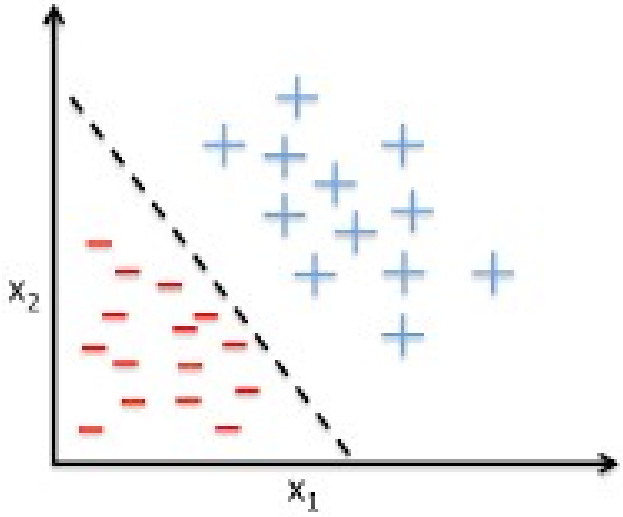


Fig. 1.2: An example of a perceptron's linear decision boundary for binary classification. Image from *Single-Layer Neural Networks and Gradient Descent* 2015.

More complicated tasks, such as classification of an image into one of a thousand different species require more complex solutions. A multi-layer perceptron uses hidden layers to capture non-linear transformations of the input. Thus, the model can handle classification problems that are not easily separated by a single line.

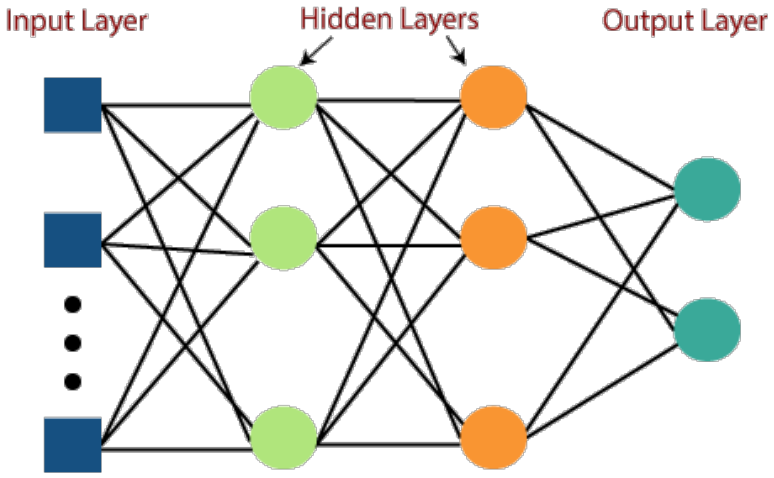


Fig. 1.3: An example of a multi-layer perceptron. Image from *Multi-layer Perceptron in TensorFlow - Javatpoint* n.d.

Here, there can be as many classification possibilites as desired. For example, if we are determining whether an image contains one of a thousand species, then the model should output 1 for the specimen that is seen and 0 for all other species. The training technique is very similar to a single layer perceptron,

except it is extended to multiple outputs using a softmax function that outputs a probability of each possible classification and chooses the highest one as the prediction. Then the model is trained using the loss between the predicted probability of each class and what probability the network should have given to each class based on the label.

1.2.3 Convolutional Neural Networks (CNN)

Multi-layer perceptrons are also called fully-connected networks because each node of adjacent layers are connected. However, having this many parameters can make training exceptionally slow. Also, it has a difficult time of finding patterns that are common of pixels in close proximity, also known as detecting local connectivity. CNNs use kernels, or filters, to find these "higher-level" features that capture patterns of pixels to be expressed in each of the hidden layers' nodes. A kernel takes input from only a subset of adjacent pixels and outputs a value for a node in the next layer. Thus, the next node is only concerned with the values of a patch of pixels rather than the whole image like in fully-connected multi-layer perceptrons. An example is shown in Figure 1.6. An example of a higher-level feature may be a face structures, as shown in 1.4. Afterwards, the outputted hidden "convolutional layer" gets further downsampled through a max-pooling layer, which effectively takes the maximum value of each subset of nodes in the convolutional layer with the assumption that the most important nodes are the ones with the highest values. An example of max pooling is shown in 1.7. More kernels are then applied to this max-pooling layer to create another convolutional layer and another max-pooling is applied. This process repeats itself as many times as desired. Training is very similar to a multi-layer perceptron, except the kernels are the model parameters θ.

1.3 Important Terminology

- **Denoising** Probably the simplest and most discussed computer vision task, the goal is to remove noise, or undesired disparity in brightness and color in an image, from an image and ultimately make it clearer. The problem is usually

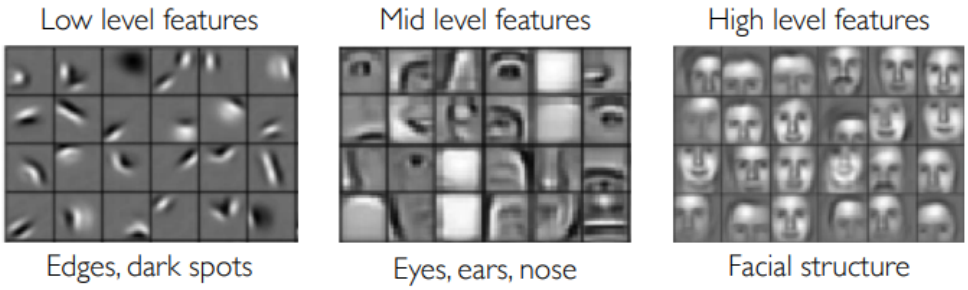


Fig. 1.4: Examples of patterns caught by hidden layer nodes. Image from Buoy 2019.

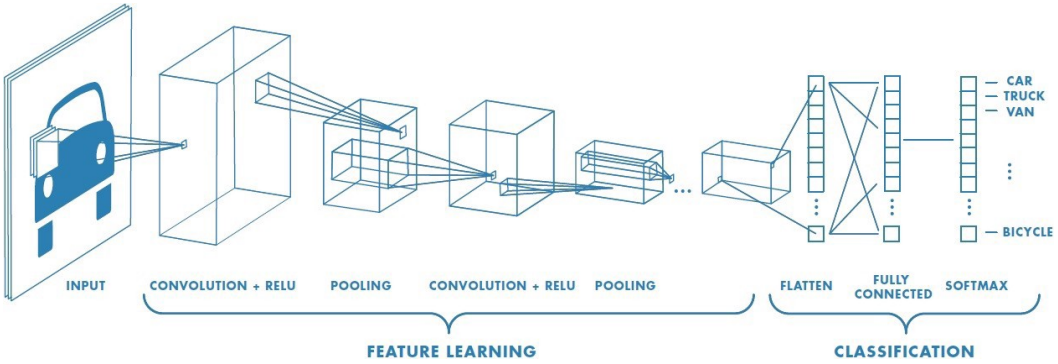


Fig. 1.5: An example of a CNN architecture. Image from Saha 2018.

formed as: $x = s + n$. Here x is our noisy image, s is the ground truth signal, and n is the added noise.

- **Peak Signal-to-Noise-Ratio (PSNR)**: The PSNR between two images represents how similar one image is to the other. A higher PSNR means the image is more similar to the other. When measuring the accuracy of computer vision results, researchers use the PSNR of various methods on the same problem to compare their success.

- **Gaussian Noise** Given $x = s + n$, where x is our noisy image, s is the ground truth signal, and n is the additive noise, we sample n from different distributions that mimic real life applications. In Gaussian noise, pixels are sampled from the normal distribution, also known as the Gaussian Distribution. Oftentimes, researchers use **white**, which is Gaussian noise where each pixel's sample is independent. The standard deviation can be tuned to widen or tighten the bell curve.

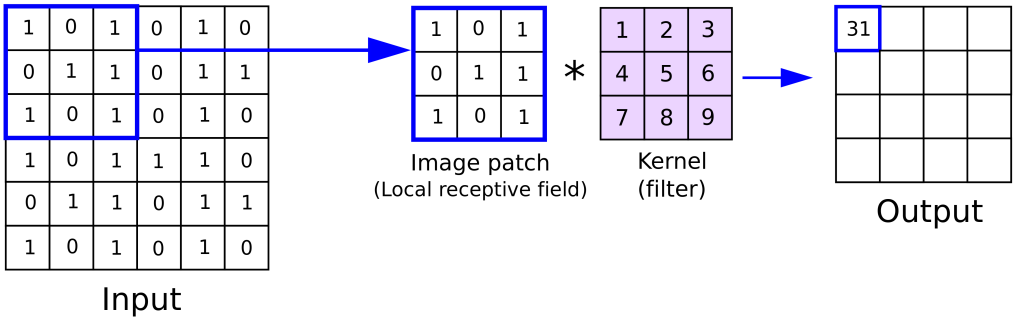


Fig. 1.6: Example of a CNN kernel. Here, the kernel is pixel-wise multiplied with the kernel to achieve the output of 31, which will be the node in the next layer. Image from Reynolds 2017.

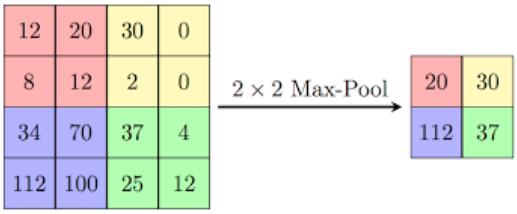


Fig. 1.7: Example of max pooling. Image from *Papers with Code - Max Pooling Explained* n.d.

Fig. 1.8: Left is noisy image, right is denoised image. Image from Burger 2012.

Fig. 1.9: Picture from * 2017.

2

Related Work

2.1 U-Net: Convolutional Networks for Biomedical Image Segmentation Ronneberger et al. 2015

Originally created for biomedical image segmentation tasks, the U-Net is an extremely popular neural network architecture used in deep learning as well as unsupervised learning methods (although not trained in the traditional sense). At the time of its creation, convolutional neural networks (CNNs) had already been successful in accomplishing image segmentation, but were slow and relied on the supervised training of huge data sets and networks with millions of parameters. However, for biomedical tasks, it is usually very difficult or impossible to even acquire a couple twenty images.

2.1.1 Architecture

U-Net first compresses the image into a feature representation. Each step in the “contracting path" repeatedly applies convolutions, rectified linear unit (ReLU) activation functions, and max pooling layers in order. Each max pooling step doubles the number of feature channels (layer thickness) and approximately halves the height and width of the layers (i.e. compression). Each step in the “expansion path" begins with an upsampling followed by a convolution (an “up-convolution") that halves the number of feature channels. *The output is then concatenated with its adjacent feature layer from the contraction phase.* This was the U-Net’s breakthrough discovery: the contraction path’s layers are used as input to their adjacent expansive path’s layers later on in the network. This allows the U-Net to use the same features used in

the compressing stage that contain the fine details and structure of the image to reconstruct it into a segmentation map. Nonetheless, the feature representation goes through two more convolutions and ReLUs after this concatenation in each step of the expansion path. See Figure 2.1 for a visualization of the network architecture.

2.1.2 Training

The training data consists of input-image segmentation map pairs. During training, a softmax is computed over all possible classes k for each pixel. The class with the highest probability from this softmax becomes the predicted label. Using stochastic gradient descent, the parameters of the U-Net change for each training image until convergence Ronneberger et al. 2015.

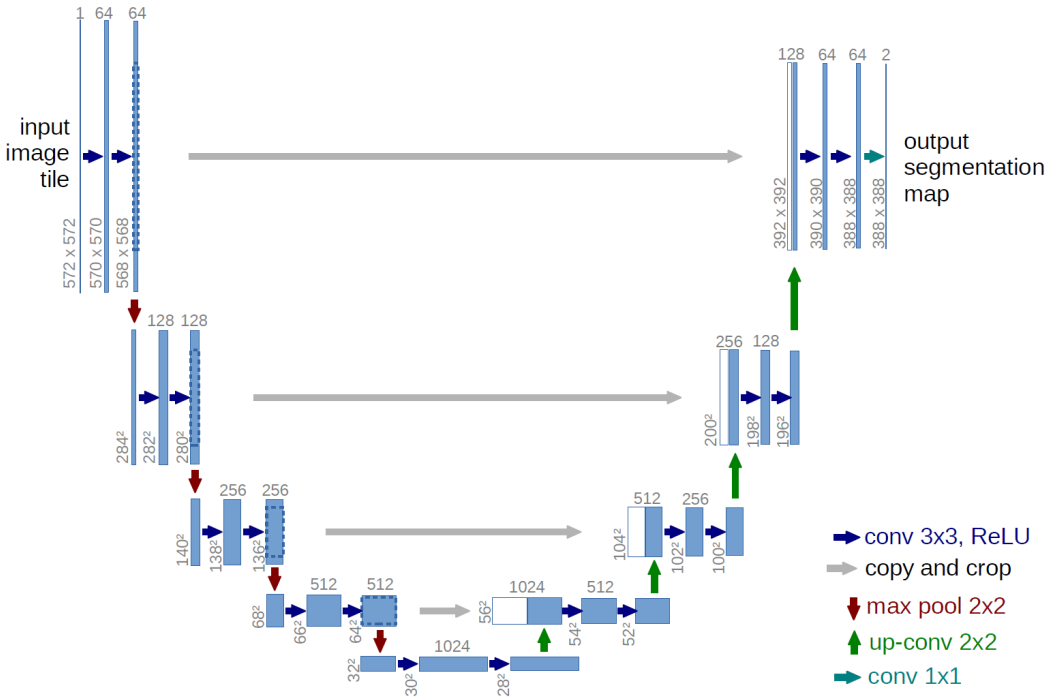


Fig. 2.1: U-Net Architecture

11

2.2 Noise2Noise Lehtinen et al. 2018

Oftentimes in machine learning, clean data is sparse while noisy data is abundant. Some examples include magnetic resonance images (MRI scans), short exposure versus costly long exposure photographs like biological studies of a sample cannot be exposed to light for long, and ray-traced renderings of a synthetic scene. Noise2Noise shows that these situations have enough dirty data to create accurate neural networks that can perform super-resolution, denoise, inpaint, and more by only using noisy-noisy data pairs rather than noisy-clean data pairs. In other words, Noise2Noise demonstrates that “we can often learn to turn bad images into good images by only looking at the bad images," and sometimes even perform better than models that use clean data. Additionally, Noise2Noise does not require an explicit image prior or distribution of the noise like DIP.

2.2.1 Theoretical Background

Traditional computer vision oriented neural networks rely on noisy data with clean labels so that we may calculate the loss and adjust our model accordingly. Such a scheme is simply outlined in Figure 4.1.

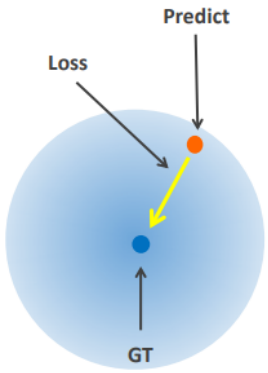


Fig. 2.2: Typical loss scheme: Use our model to make a prediction, compare it with the ground truth to calculate the loss, change our model accordingly. Image from Lee et al. n.d.

However, noisy images are still very close to the ground truth, and, on average, many noisy images will average out to the signal as long as the noisy images are sampled from the same zero-mean distribution. For example, “a long, noise-free exposure is the average of short independent, noisy exposures." Thus, if we have

many noisy images with the same subject matter, we can train a neural network with (noisy, noisy) instead of (noisy, clean) data points. The loss from each data pair will be correct on average, as outlined in Figure 2.4.

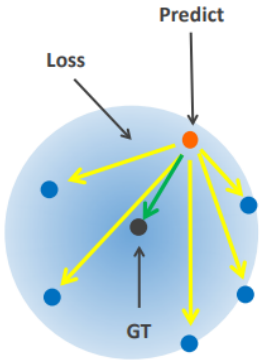


Fig. 2.3: N2N novel training scheme: Use our model to make a prediction, compare the prediction with another noisy image to calculate the loss (which will be correct on average), change our model accordingly. Image from Lee et al. n.d.

2.2.2 Experiments

Denoising With denoising, test convergence speed for Noise2Noise is actually mirrors that of supervised methods, as shown in Figure 2.4. Further, if one only has a finite amount of resources available to capture an image, or a “fixed capture budget," using Noise2Noise actually produces *better* results. For example, say one has 2000 capture units (CUs) or capture resources, and each noisy piece of data costs 1 CU and each clean piece of data costs 19 CU. The authors found that seeing more noisy images and seeing images of different subject matter is the best way to spend ones capture budget. These findings are exhibited in Figure 2.5.

MAYBE ADD DIP???

2.3 Noise2Void Krull et al. 2018

Noise2Noise distinguished the need for clean data collection by only using noisy image pairs as data. However, even noisy data can be difficult to obtain or scarce, especially biomedical data. Noise2Void aims to remove the reliance on data pairs at

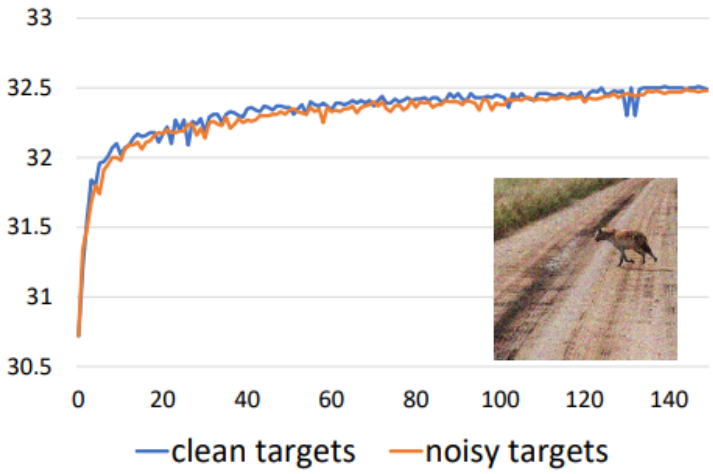


Fig. 2.4: Convergence speed of supervised learning with clean targets versus N2N, which only uses noisy targets.

all by training CNNs “directly on a body of noisy images," or, similar to DIP, just on the single image to be denoised.

2.3.1 Training

Instead of training on (noisy, noisy) image pairs, the authors were able to train a neural network on just one image at a time by selecting patches of pixels known as “receptive fields” that exclude the central pixel in the patch and use it as a target, as shown in Figure 2.6. Such a network was termed a “blind-spot network." In Noise2Void, the U-Net is used. For each patch, the loss used for training is the mean squared error (MSE) between the target pixel s_i^j and the model's prediction for the pixel $\hat{s}_i^j$:

$$\mathcal{L}(\hat{s}_i^j, s_i^j) = (\hat{s}_i^j - s_i^j)^2 \tag{2.1}$$

Although Noise2Void can be trained with one noisy image, it performs better when using multiple noisy training images of the same subject matter like Noise2Noise. In these situations, the optimization problem aims to minimize the loss between each pixel i's input value and predicted value among every training image j:

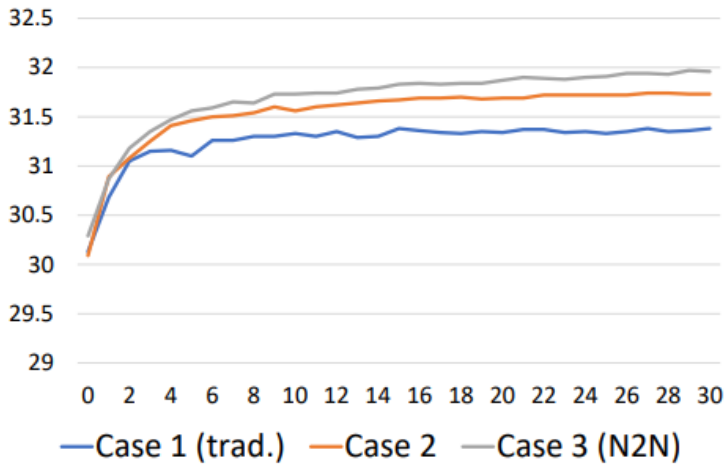


Fig. 2.5: In order of increasing accuracy, Case 1 spends its budget on 100 noisy-clean data pairs each costing 20 CUs (1 CU for noisy image and 19 for the clean), Case 2 is the same as Case 1 except each noisy image is also paired off with every clean image for extra data points (a total of 100 * 20 * 19 = 38000 training pairs), and Case 3 spends its capture budget on just two noisy-noisy images for each of 1000 different subject matters.

$$\underset{\theta}{\operatorname{argmin}} \sum_{j} \sum_{i} (\hat{s}_i^j - s_i^j)^2 \tag{2.2}$$

Additionally, the authors produce augmented data of the noisy images by adding by adding noise and flipping the picture 90 three times.

2.3.2 Results

The authors highlight that Noise2Void is not expected to outperform traditionally trained denoising networks, but usually falls within only one or two dB as shown in Figure 2.7. The main advantage is Noise2Void's ability to learn with only a single image, which is not possible for traditional supervised learning or Noise2Noise. One example shown in Figure 2.7 displays fluorescence microscopy data that has only one noisy image and no ground truth available. Although it performs a bit less accurately than other leading single-image denoisers such as BM3D, it takes only a fraction of the runtime.

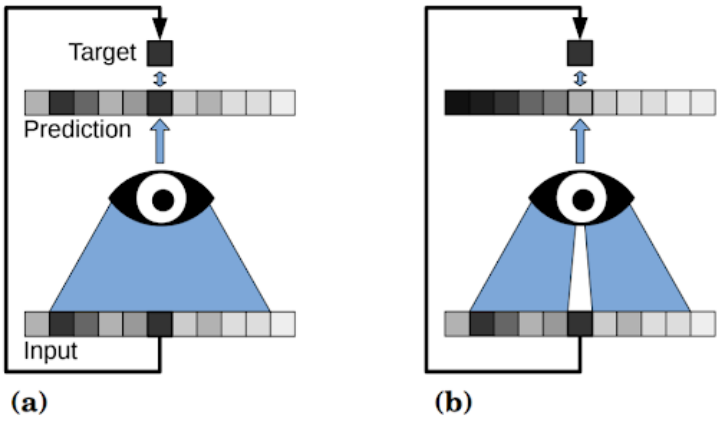


Fig. 2.6: Here are examples of patches or “receptive fields." Note if authors did not exclude the central pixel, the networks prediction for the pixel would just be what it saw in the original image, thus training a useless identity network, as shown in **(a)**. In **(b)**, Noise2Void’s receptive field excludes the central pixel and trains the network by using the excluded pixel’s original value as the target.

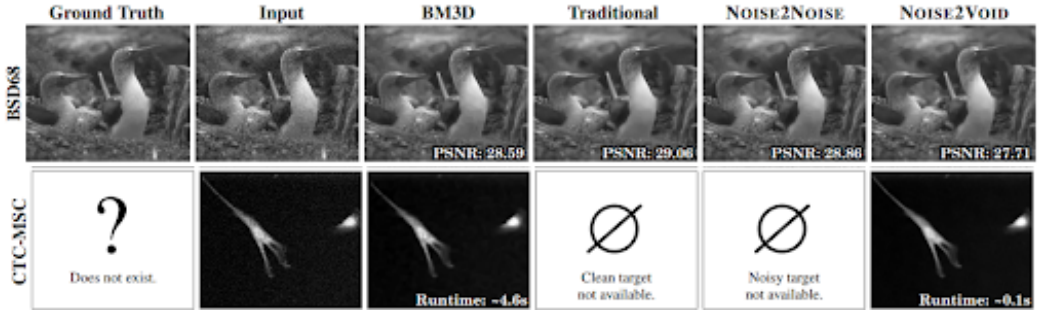


Fig. 2.7: On top: average PSNR for the BSD68 dataset for related algorithms. Bottom is fluorescence microscopy data. No ground truth is available, so PSNR could not be measured for the methods that require pairs: Traditional and Noise2Noise.

2.3.3 Theory

The intuition behind this method is, although the target pixel may be noisy because the input image is noisy, it will still represent ground-truth signal on average. Thus, the network will be trained correctly. A very similar method called **Noise2Self** offers a mathematical explanation for this method. Say x is our noisy label, $\hat{x}$ is the noisy label input with excluded pixels, and y is the ground truth (that is often unavailable). Taking the expectation of the objective function $||f(\hat{x}) - x||^2$ and using typical assumptions of mean-zero noise $\mathbb{E}[x|y] = y$ and conditional independence between x and $\hat{x}$ given y, it can be shown the following is true:

$$\mathbb{E}||f(\hat{x}) - x||^2 = \mathbb{E}||f(\hat{x}) - y||^2 + \mathbb{E}||x - y||^2 \tag{2.3}$$

That is the expected value of the noisy label loss $||f(\hat{x}) - x||^2$ is exactly equal to the expected value of the clean label loss $||f(\hat{x}) - y||^2$ with some additive noise variance. Thus, minimizing the loss of between our network's prediction $f(\hat{x})$ and the noisy label x *is the same* as minimizing the loss between $f(\hat{x})$ and the ground truth y (up to a constant noise variance term $(x - y)$ that is independent of the function f).

17

3

Outline

3.1 Statement of Problem:

As described above, there is a wide need for methods like N2V that solve computer vision tasks with little to no data. Recently, many methods have been discovered that allow competent self-supervised training on noisy images that almost match state-of-the-art supervised methods' results. One such powerful method, N2V, demonstrates the effectiveness of a scheme that uses one pixel as a target in its receptive field. However, we hypothesize that this method is subject to overfitting because it only uses the target's neighboring pixels to predict. We aim to test whether or not a small group of pixels, or a "subpatch," can be used as the target instead of a single pixel to improve the model. A model that utilizes subpatches could be beneficial because its network is able to pick up on features that can only be shared between subpatches and not pixels (for example, an edge), and use them to generalize to future predictions of subpatches that have similar qualities. Another prominent question our research seeks to answer is whether a "random subpatch-weight scheme" can enhance results. In such a scheme, random subpatches from around the image are used as the receptive field during training. The randomly-selected subpatches in this new receptive field are weighted differently based on their proximity and color during training: a patch that is closer in color or closer in proximity to the target subpatch is assigned a higher weight. This could further improve the model because the network is able to learn from all over the image rather than just the local pixels around the target as in N2V, thereby increasing generalization and preventing overfitting.

3.2 Methods

3.2.1 Noise2Patch Training

We adapt N2V's blind-spot network to a "blind-subpatch network" in which random subpatches from the entire image are arranged around the central, target subpatch. To align our language with the explanation of our experiment, we will refer to each of these patches as *subpatches* that form the *larger patch* that is our receptive field. For example, say we select a subpatch size of 10×10 pixels to be our target subpatch and eight other randomly selected 10×10 subpatches to be our predictive subpatches. We would arrange these subpatches into a larger 30×30 patch that has three rows and three columns of subpatches with the censored patch in the center.

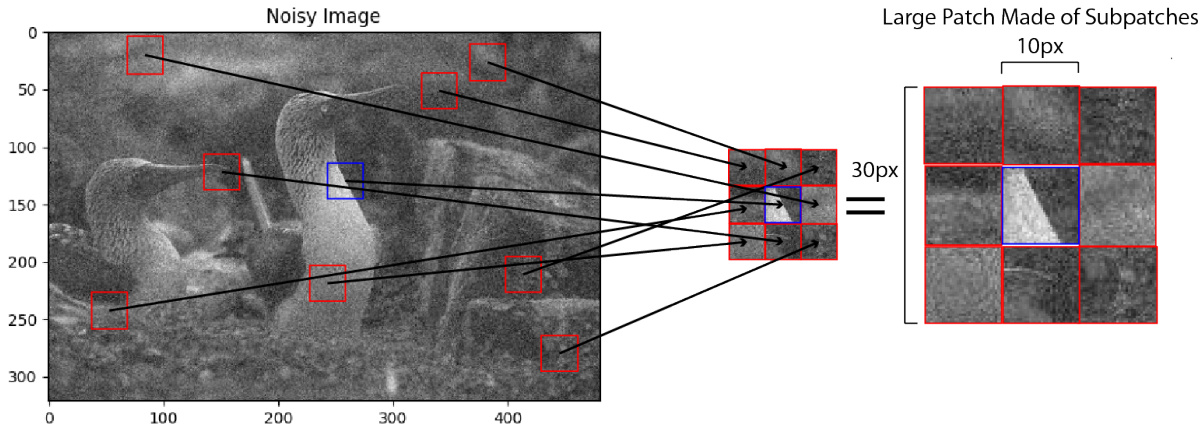


Fig. 3.1: N2P compiles many subpatches into a large patch. One possible structure is a 9×9 grid. The "censored," central subpatch outlined in blue is the target patch the network is trying to predict during training. The random subpatches outlined in red are given to the network to aid in the prediction of the target patch.

Furthermore, each of i subpatches has a weight s_i associated with it based on the difference in color and positioning from the target subpatch. The weight is calculated as follows:

$$s_i = e^{-\left(\frac{1}{\sigma_I^2}||I_i - I_j||_2^2 + \frac{1}{\sigma_X^2}||X_i - X_j||_2^2\right)} \tag{3.1}$$

19

Here, I_i and I_j are the average colors of subpatch i and target subpatch j respectively. X_i and X_j are the average positions (just the position of each subpatch's middle pixel in practice) of subpatch i and target subpatch j respectively. σ_I and σ_X are hyperparameters to regulate the importance of color and positioning in the weight function. Through trial and error our experiments we set σ_I and σ_X. To apply the weights, we attach a weight-channel (another layer to the large patch) to the input that is the same shape as the large patch. Thus, each pixel p_k in the large patch pertains to a single weight w_k in the weight channel. Each w_k is equal to its respective subpatch's weight. Note we considered having a different weight for each pixel rather than having one weight for all pixels in a given subpatch, but chose the latter because it gives the network a better understanding of the input's subpatch structure when there are clear boundaries of subpatches in the weight channel. Additionally, this method improves efficiency of an already computationally expensive task by calculating one weight per subpatch of pixels rather than a weight for every pixel.

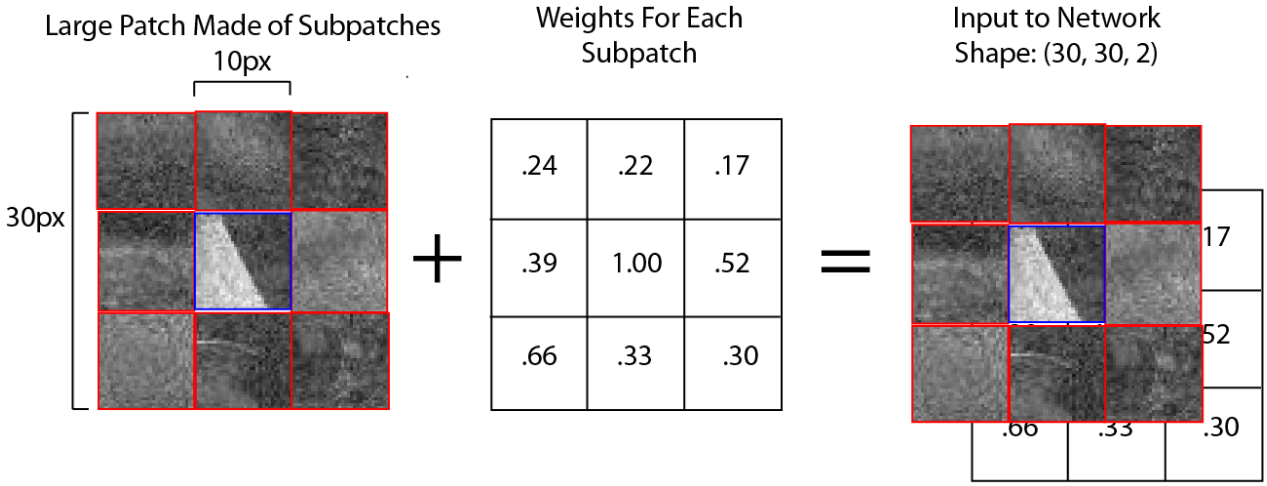


Fig. 3.2: Weights are calculated for each subpatch in the compilation that makes up the large patch described above. A weight channel of the same shape as the large patch is added "behind" the large patch. Each pixel in the weight channel contains the weight of its corresponding pixel's subpatch. Note that the central patch is actually censored. The weight of 1.00 is symbolic to show that the target subpatch has exactly the same color and position in the original graph as itself.

N2V trains by minimizing the difference between patch i of image j and its "receptive field" (the same patch with pixels censored):

$$\underset{\theta}{\operatorname{argmin}} \sum_j \sum_i L(f(x^j_{RF(i)}; \theta), x^j_i) \tag{3.2}$$

20

Here, x_i^j is the patch i of image j and $x_{RF(i)}^j$ is the patch's receptive field. N2P uses the same objective function, except x_i^j is instead the "mosaic" of subpatches built as described above, and its receptive field $x_{RF(i)}^j$ is everything in x_i^j without the censored patch.

3.2.2 Implementation

We adapt N2V's code to fit our scheme. For each patch, we sample the desired amount of subpatches and fit them into the larger patch rather than just sampling one large patch as in N2V. N2V actually censors N pixels scattered around each patch for efficiency. We take advantage of this strategy by making those N pixels exactly the ones in the central subpatch we want to censor [Krull et al. 2018].

In practice, N2V uses a "fake" color value for the censored pixels in $x_{RF(i)}^j$. For each censored pixel, its fake value is a random neighbor in the pixel's close vicinity. This will not work in N2P because some censored pixel's neighbors include pixels from other subpatches that could be drastically different colors, while some have just other censored pixel as neighbors. Thus, we instead use N2V's built in option for additive noise, which adds a random number to the original pixel value sampled from a Gaussian distribution with zero-mean and hyperparameter σ, effectively making each censored pixel close to the same value, but not quite. This is very similar to the outcome of using a random pixel that is within the vicinity of the censored pixel. Additionally, having a color similar (but not the identity because the network would just output the input in that case) is actually desirable because the subpatch in question's noisy values should be useful in predicting its ground truth values. Note this logic is also reflected in the fact that the censored subpatch's "fake pixels" will have the largest weight as shown in equation 3.1.

To predict, N2P splits the test image into subpatches and predicts each subpatch seperately to form the prediction for the entire image. Again, we compile random subpatches from the image around the target and predict just the central, target subpatch. Finally, we reassemble the subpatch predictions into the original image's orientation.

Lastly, N2V uses a library developed from some of the same authors of N2V called content aware image restoration (CARE) [Weigert et al. 2018], which adapts the

U-Net architecture for image restoration in the context of fluorescence microscopy data. N2V generally leaves this software untouched, and we do the same.

3.3 Experiments

For all experiments up to section 3.3.4, we use Intel Core i7-8550U CPU. For experiments thereafter, we use an NVIDIA GP104GL GPU. We conduct experiments on the same 400 180×180 gray scale images as N2V. We also test on the same gray scale version of the BSD68 dataset, adding zero mean Gaussian noise with a standard deviation of 25 to create noisy data. For simplicity of experiments, we keep most training parameters the same as well unless otherwise specified: U-Net with depth 2, kernel size 3, batch normalization, a linear activation function in the last layer, 96 feature maps in the initial level, learning rate of 0.0004, batch size of 128, and 200 training epochs.

3.3.1 Initial Experiments

Our initial experiments have a larger patch size of 64 pixels $\times$ 64 pixels the same. We choose a subpatch size of 8 pixels $\times$ 8 pixels for an 8 subpatch $\times$ 8 subpatch grid of subpatch within the larger patch. Finally, we increase the standard deviation of the noise added to the target subpatch from 2 to 5 because randomly chosen subpatches will be drastically different from the target in our random-subpatch network (as opposed to the N2V network where the target's neighboring pixels in the larger patch are likely to be similar), so we do not want to overfit to the target subpatch's "fake" pixel values.

In Figure 3.4, we compare Noise2Patch with other methods. The PSNR of the original noisy input image is 20.00dB. With the initial parameters described above our method actually adds noise, thereby decreasing the PSNR to 19.45dB. In 3.3, we observe a drastic decrease in the average training loss per patch in the first few epochs of training and only a moderate decrease in the validation loss (validation loss is the average loss per patch on a small reserved validation set). This indicates overfitting.

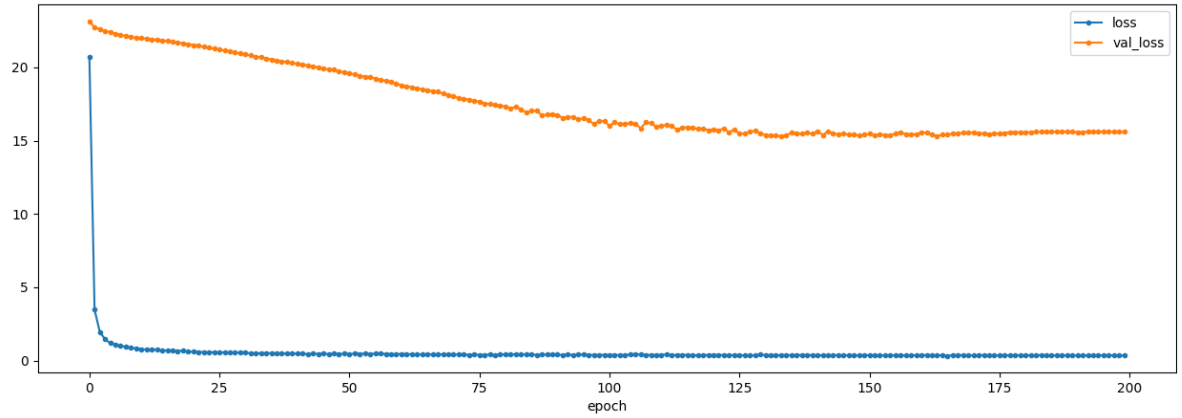


Fig. 3.3: Loss during training of our initial N2P configuration. The blue line represents the average loss per training patch in each epoch of training (loss calculation described in 3.2). The orange line represent the average loss per patch in a small held out validation set. The loss drastically decreases in the first few training iterations, indicating overfitting.

3.3.2 Preventing Overfitting Experiments

Increasing Target Noise

The most likely source of overfitting is that the network learns that random subpatches are usually not helpful in determining the true values of the target subpatch, so it just uses the "fake" values of the input target subpatch as its best guess. One way to mitigate the network overusing the fake values of the target subpatch's pixels is to increase the additive noise of these fake values. The implementation consists of increasing the standard deviation of the additive Gaussian noise by varying amounts.

As shown in 3.5, adding more noise to the target to "block it out" does not improve the model, and, while 3.6 shows that there is slightly less overfitting, learning seems to halt and plateau around only 25 training iterations.

Decreasing The Batch Size

The second most likely source of overfitting is having such a small, limited training set. We could mitigate the network overfitting to the training data by decreasing the

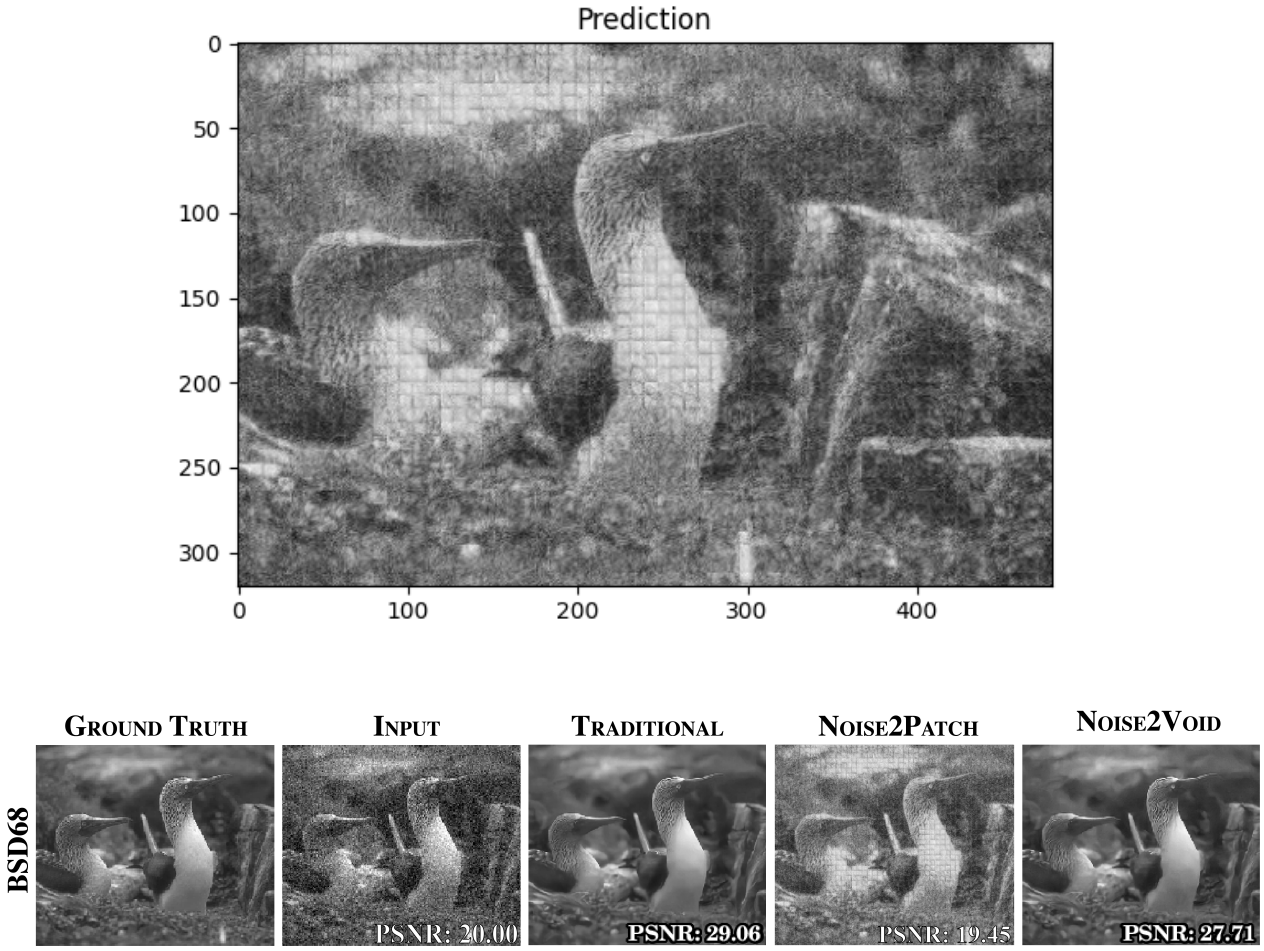
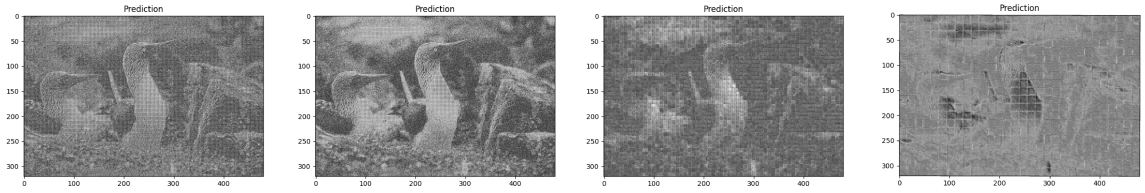


Fig. 3.4: Comparing the results of a traditional supervised CNN and Noise2Void with N2P. N2P creates a sort of mosaic looking prediction and actually ends up adding more noise to the input image, thereby decreasing the PSNR value. This particular N2P configuration was comprised of 64 px $\times$ 64 px patches containing 8 px $\times$ 8 px subpatches, additive Gaussian noise of standard deviation 5, batch size of 128, and 200 epochs.

24



Standard Devation:	10	20	50	75
PSNR:	18.71	19.40	17.57	16.56

Fig. 3.5: Testing varying amounts of additive Gaussian Noise in the target subpatch. Generally, adding more noise equates to a less accurate prediction.

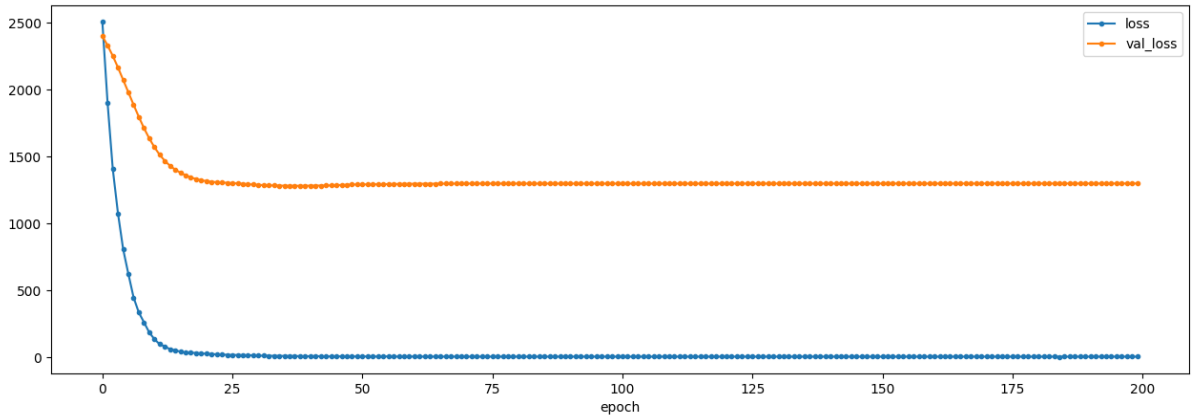


Fig. 3.6: The loss graph for a standard deviation of additive noise of 75. Meaning of the graph's components described in 3.3. Although, the graph's training loss (blue line) has a less steep incline indicating less outfitting, the validation loss plateaus at a much higher value than in our initial experiment.

batch size so that network updates are noisier, which could induce regularization making the network less susceptible to overfitting. Here, we test alternative batch sizes of 1 (so updating after every training example) and 32. We also decrease the amount of epochs to 40 and 80 respectively to make up for a drastic increase in runtime due to increased updates to the network (the network is updated after each batch, so a smaller batch size results in more updates).

As shown in 3.7, decreasing the batch size to 32 had little effect on results, and according to 3.8, overfitting is not prevented. Moreover, decreasing the batch size to an extreme amount of 1 drastically hinders results.

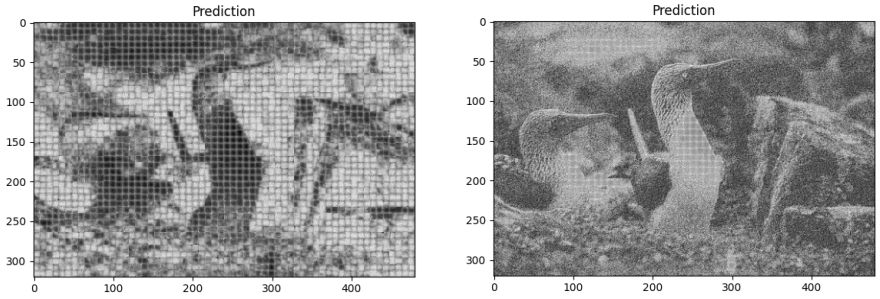
Only Color Weighted Experiments

Only using color to determine a random subpatch's weight in predicting the target subpatch could prevent overfitting because we are placing more importance on subpatches from the entire image rather than just subpatches in the vicinity of the target, thereby strengthening the model's generalization. Additionally, we suspect a random subpatch's color is likely more important than its proximity in determining its weight because two nearby subpatches can have drastically different colors. Thus, we experiment with taking position out of the equation and only using a random subpatch's similarity in color to the target subpatch to calculate its weight.

From 3.10, we can tell that overfitting is still taking place, although the training loss doesn't plateau quite as quick as in the initial experiment. Nevertheless, the accuracy decreases to 18.84dB.

3.3.3 Different Patch and Subpatch Sizes

Larger patch sizes give more subpatches for the network to use to predict the target subpatch, and could therefore be beneficial. However, having too many subpatches could make it difficult for the network to learn what exactly makes a subpatch important (a higher weight) by diluting the network with so many parameters. Additionally, processing larger patches would be more computationally expensive. Larger subpatches could allow the network to learn broader features, but also miss out on finer details. Here, we test varying combinations of patch and subpatch sizes to hone in on an optimal setting.



Batch Size:	1	32
PSNR:	13.42	19.45

Fig. 3.7: Testing varying batch sizes. Batch sizes close to our original experiment have little to no effect on results. Using an extremely small batch size adds a significant amount of noise.

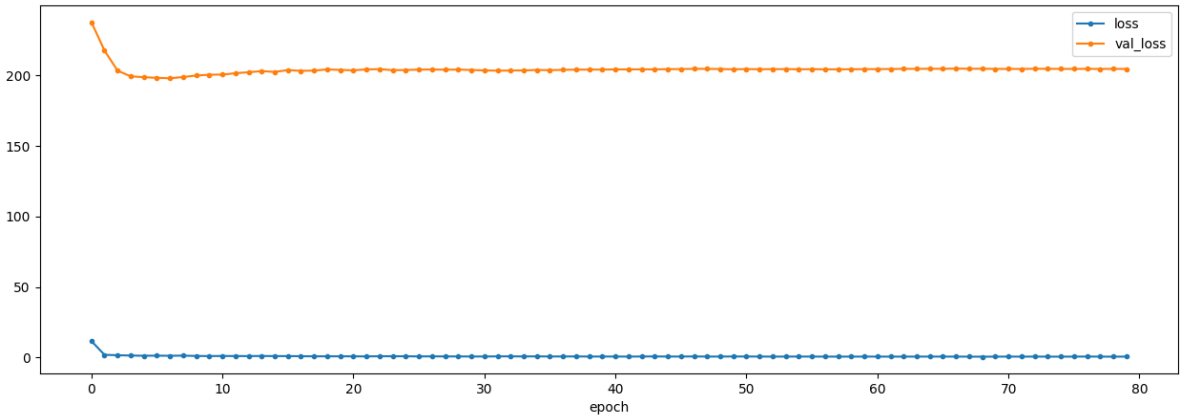


Fig. 3.8: The loss graph for a batch size of 32. Meaning of the graph's components described in 3.3. The initial training loss is much smaller and plateaus almost immediately. The validation loss starts out larger and plateaus quicker, but with a gradual increase over the entire training period.

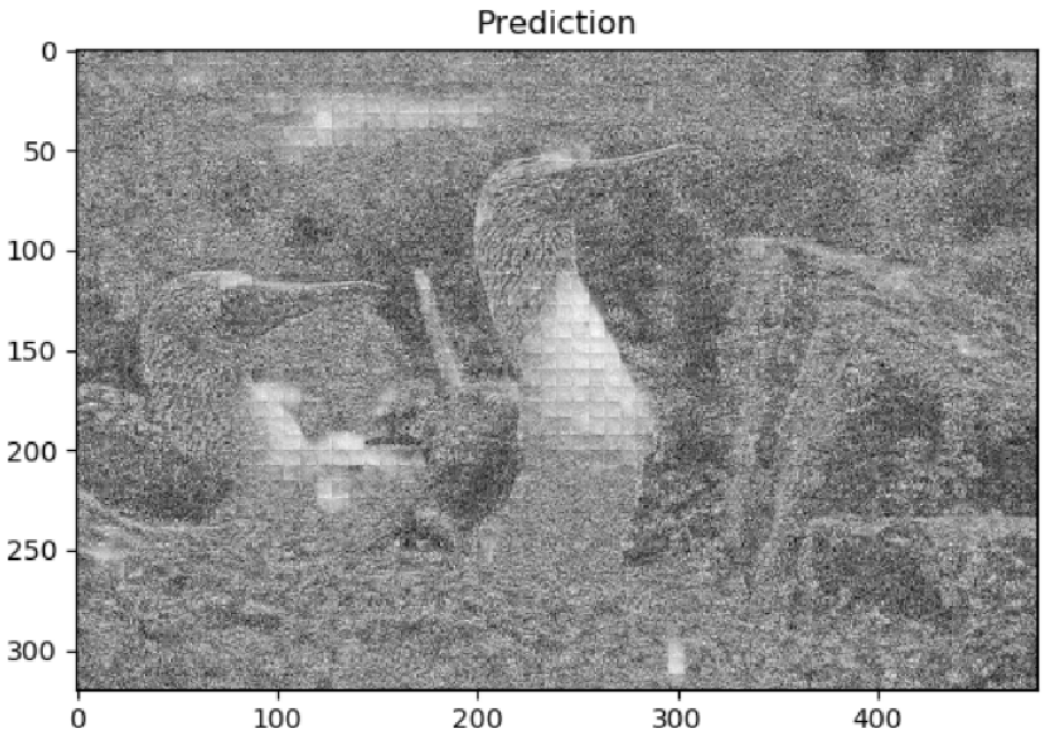


Fig. 3.9: Prediction of only using color-similarity to produce the weight of a subpatch, achieving an average PSNR of 18.84dB.

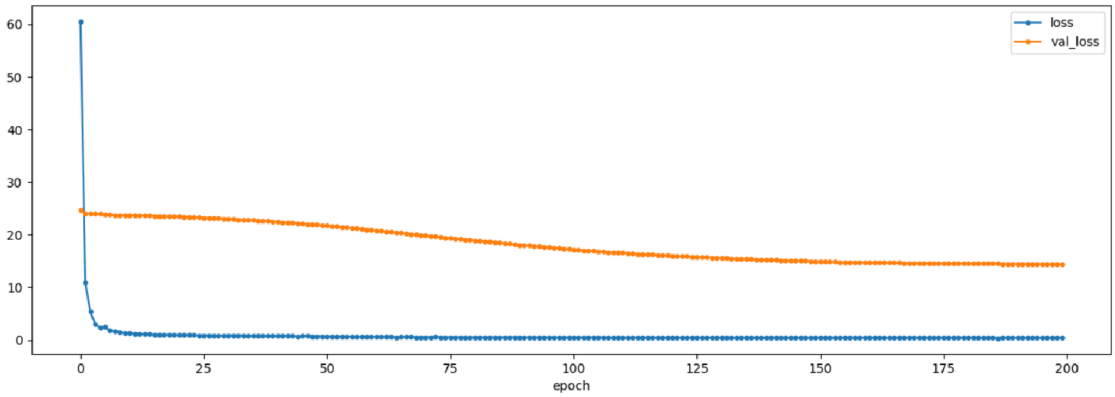
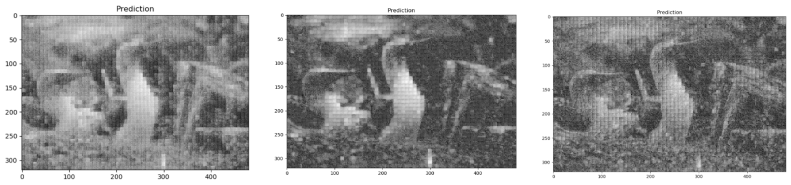


Fig. 3.10: The loss graph for a model that only uses color in the calculating the weight of equation 3.2. Meaning of the graph's components described in 3.3. Not much changed from the initial experiment except the starting point of the training loss. Thus, we still suspect overfitting.

28



Patch Size:	16	32	128
PSNR:	19.61	22.05	20.31

Fig. 3.11: Average PSNR based on varying patch sizes. Larger patch sizes equate to more random 8 px $\times$ 8 px subpatches the network can use to predict the target subpatch. For example, a 16 px $\times$ 16 px patch has 4 subpatches including the target whereas a 128 px $\times$ 128 px patch has 256.

Varying Patch Size

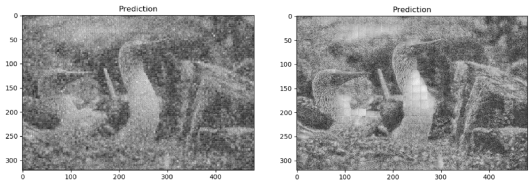
Keeping all parameters including subpatch size the same, we test a larger patch size of 128 allowing for 256 subpatches instead of 64, and a smaller patch size of 32.

Increasing the patch size to 128 does marginally increase the accuracy of the model to 20.31dB, likely because of the additional random subpatches to base the target subpatch's prediction off of. Decreasing the patch size to 32 increases the accuracy significantly to 22.05dB, likely because the network can better gain an idea of the subpatch structure with less parameters to follow. Yet, decreasing even further to a patch size of 16 likely has the reverse effect of having too little subpatches to base the target subpatch's prediction off of.

Varying Subpatch Size

Again keeping all parameters the same, we test the effectiveness of using a larger subpatch size of 16 px$\times 16$ px and smaller subpatch sizes of 4 px$\times 4$ px and 1 px$\times 1$ px (pixel-level subpatches). Note that using a subpatch size of 1 is essentially N2V except we use weighted random pixels instead of just neighboring pixels.

Raising the subpatch size to 16 px $\times$ 16 px marginally increases results, while shrinking to 4 px$\times 4$ px decreased results. Unfortunately, we were unable to test 1 px$\times 1$ px due to time and memory constraints. This could be a source of future exploration.



Subpatch Size:	1	4	16
PSNR:	NA	18.47	20.48

Fig. 3.12: Average PSNR or test set for varying subpatch sizes (all subpatches are square). Unfortunately, we were unable to test a subpatch size of 1 px $\times$ 1 px due to runtime and memory constraints, but we speculate that size could enhance results greatly.

3.3.4 Nearby Subpatch Experiments

Nearby Subpatches for Testing

As described in 3.2.2, N2P predictions are made by dividing the images into individual subpatches and arranging random subpatches with associated weights around it to create an input patch. The predictions for each of these patches are then recompiled into their original positions in the image to form the prediction of the entire image. However, it is not necessary to use random subpatches during prediction. Similar to N2V, using the subpatches that neighbor the target in the original image would likely be benefit the prediction because these adjacent subpatches should be more alike to the target, and thus more useful. Note, these neighboring subpatches will usually have a high weight because it is highly likely that neighboring subpatches have a very similar color to the target subpatch.

For this experiment, we use the same random subpatch scheme and parameters as in our intial experiments. 3.13 shows an example of using the target subpatch's immediate, adjacent subpatches for prediction. Using nearby subpatches during testing actually increases the average PSNR of the test set predictions to 21.12. However, the image does look blurry and ultimately more noisy than before. Therefore, this may not be desirable even though the image has been "denoised" a small amount.

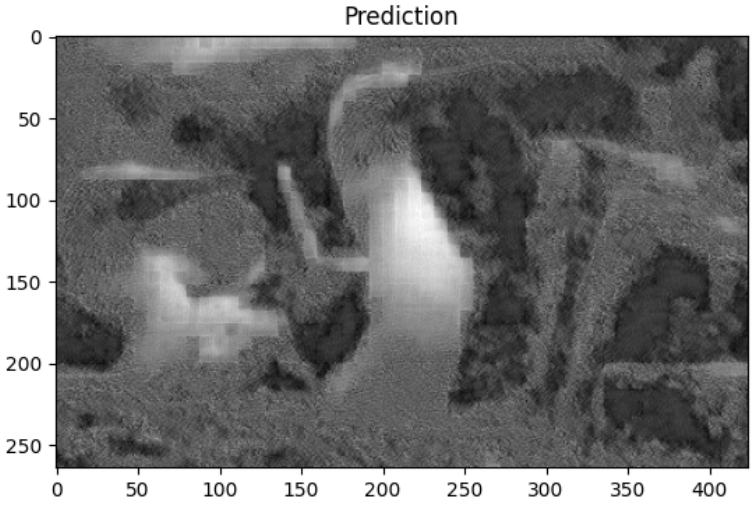


Fig. 3.13: Training for this model uses the random subpatch scheme and parameters of our initial experiments. However, predictions are made using the target subpatch's neighboring patches rather than random subpatches. This strategy achieves an average PSNR of 21.12.

Nearby Subpatches for Training and Testing

We can not only use nearby subpatches for testing as in 3.3.4, but also for training. This scheme is very similar to N2V, which uses local, neighboring pixels to predict the patch's target pixel. Here, we use local subpatches to predict the target subpatch.

In 3.14, the familiar traces of blocky subpatches in the prediction are almost gone. A small amount of denoising took place for an increase to 22.07dB, which was expected due the "nearby method's" similarity to N2V.

3.3.5 Longer Runtime

Based on our experiments above, we choose a combination of high performing strategies to use in a longer training period (more epochs): using nearby subpatches during training and testing with a larger patch of size 32 px $\times$ 32 px.

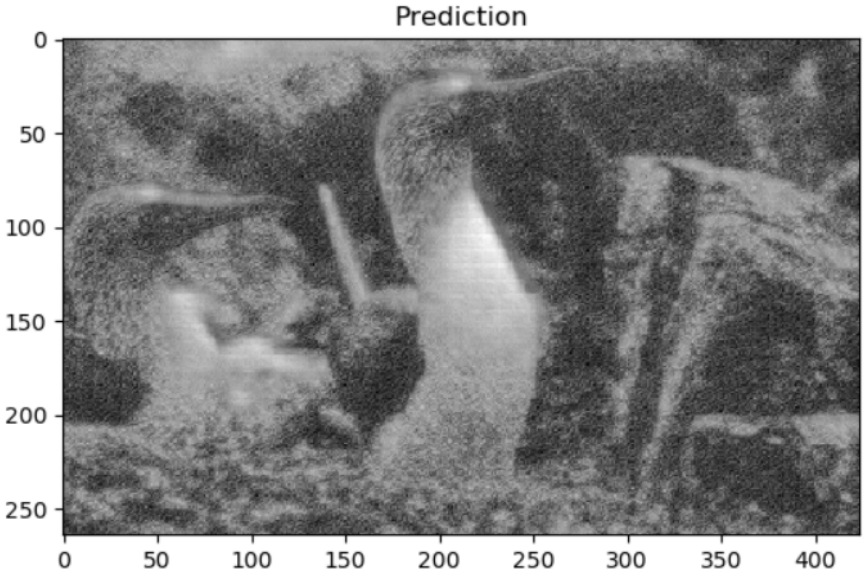


Fig. 3.14: Instead of using random subpatches to dictate the target subpatch, we use the neighboring subpatches during training and testing, very much like N2V. This procedure achieves an average PSNR of 22.07dB.

32

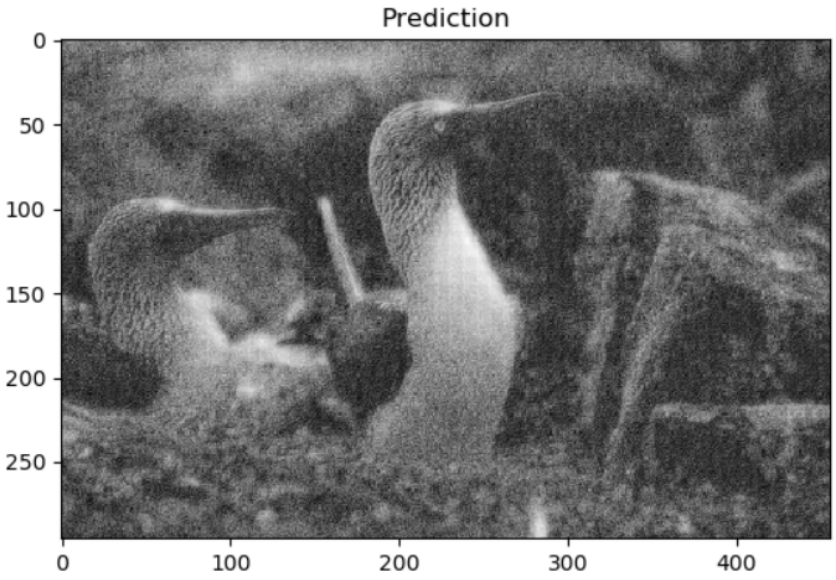


Fig. 3.15: Example prediction of using our chosen optimal setting of 32 px $\times$ 32 px patch sizes with the N2V-like nearby scheme described in 3.3.4. Combining these procedures achieves an average PSNR of 17.75dB of the test set.

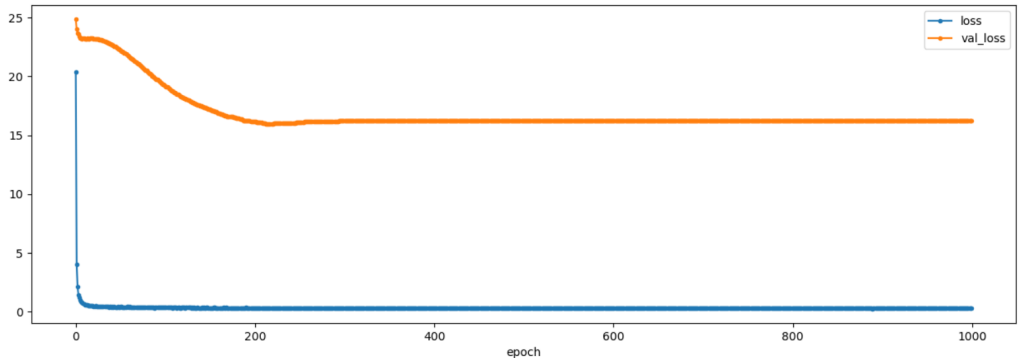


Fig. 3.16: The loss graph for our chosen optimal setting of 32 px $\times$ 32 px patch sizes with only nearby subpatches during training and testing. There is not much difference in the graphs between our “optimal setting" and our initial experiment. Both graphs’ learning seems to plateau very early.

Our optimal setting does not perform as expected and actually “adds” noise for a drop in average PSNR to 17.75dB.

4

Conclusion and Future Work

Here we present N2P, a novel unsupervised deep learning method for denoising. N2P does not enhance denoising results beyond N2V by preventing overfitting. Thus, we speculate that N2V does not overfit. However, this claim should be taken lightly because our efforts to prevent overfitting in our own model were not very successful as described in 3.3.2. Additionally, we have found evidence that such a scheme does not denoise well, only achieving a maximum of 22.07dB in our experiments. One of the most obvious defects of our predictions is that subpatches of pixels are easily visible, creating an undesirable "blocky" ouput that is likely caused by N2P's enforcement of a subpatch structure. We conjecture this is because the network is unsure of what lies at the edges of each subpatch due to the absence of information about the pixels around the edge. However, using nearby subpatches solved this issue as shown in 3.14.

Finally, a notable limitation was our lack of training iterations due to extensive training times. We did not have access too GPUs for much of our research period, so this hindered our ability to train beyond 200 epochs. Thus, future work could involve running similar experiments for longer.

One aforementioned avenue of future work would be a sort of combination of N2V and N2P: using the random subpatch scheme on subpatches of size 1 px $\times$ 1 px. Runtime and memory constraints prohibited us from performing such experiments, but we believe it could improve results by copying a N2V-type procedure, which has already been proven to be successful, while allowing for learning from the entire image through the random subpatch strategy.

Another direction of future work could actually just be a revised version of N2P where the input consists of subpatches that are compiled on top of each other rather than the mosaic formation in N2P.

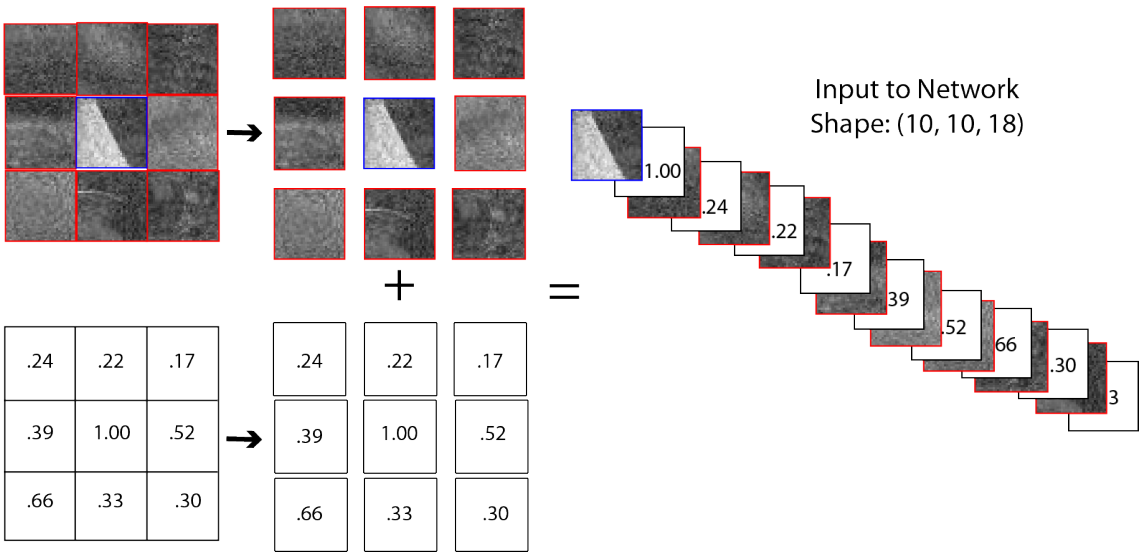


Fig. 4.1: N2P compiles many subpatches into a 9×9 large patch. The "censored," central subpatch outlined in blue is the target patch the network is trying to predict during training. The random subpatches outlined in red are given to the network to aid in the prediction of the target patch.

This method solves one of the major drawbacks of N2P: the network has to actually learn to discern the subpatch structure in the compilation of the larger patch. Such a method would remove the need to learn subpatch boundaries at all by having a different subpatch for each layer.

References

*, Name (June 2017). *Amund Tveit's Blog*. URL: https://amundtveit.com/2017/06/04/deep-learning-for-image-super-resolution-scale-up/ (cit. on p. 9).

Buoy, Rina (Dec. 2019). *Convolutional Neural Network-An Informal Introduction (Part-1)*. URL: https://medium.com/analytics-vidhya/convolutional-neural-network-an-informal-intro-part-1-db9fca86a750 (cit. on p. 7).

Burger, Harold C (Nov. 2012). URL: http://people.tuebingen.mpg.de/burger/neural_denoising/ (cit. on p. 8).

Krull, Alexander, Tim-Oliver Buchholz, and Florian Jug (2018). "Noise2Void - Learning Denoising from Single Noisy Images". In: *CoRR* abs/1811.10980. arXiv: 1811.10980. URL: http://arxiv.org/abs/1811.10980 (cit. on pp. 13, 21).

Lee, Cheolin, Minki Jo, and Nick Heppert (n.d.) (cit. on pp. 12, 13).

Lehtinen, Jaakko, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila (2018). "Noise2Noise: Learning Image Restoration without Clean Data". In: *CoRR* abs/1803.04189. arXiv: 1803.04189. URL: http://arxiv.org/abs/1803.04189 (cit. on p. 12).

Multi-layer Perceptron in TensorFlow - Javatpoint (n.d.). URL: https://www.javatpoint.com/multi-layer-perceptron-in-tensorflow (cit. on p. 5).

Papers with Code - Max Pooling Explained (n.d.). URL: https://paperswithcode.com/method/max-pooling (cit. on p. 8).

Quan, Yuhui, Mingqin Chen, Tongyao Pang, and Hui Ji (2020). "Self2Self With Dropout: Learning Self-Supervised Denoising From Single Image". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1887–1895. DOI: 10.1109/CVPR42600.2020.00196 (cit. on p. 3).

RainerGewalt and Name * (Mar. 2021). *Perceptrons - These Artificial Neurons Are The Fundamentals Of Neural Networks*. URL: https://starship-knowledge.com/neural-networks-perceptrons (cit. on p. 4).

Reynolds, Anh H. (Oct. 2017). *Convolutional Neural Networks (CNNs)*. URL: https://anhreynolds.com/blogs/cnn.html (cit. on p. 8).

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. cite arxiv:1505.04597Comment: conditionally accepted at MICCAI 2015. URL: http://arxiv.org/abs/1505.04597 (cit. on pp. 10, 11).

Saha, Sumit (Dec. 2018). *A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way*. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (cit. on p. 7).

Single-Layer Neural Networks and Gradient Descent (Mar. 2015). URL: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html (cit. on p. 5).

Ulyanov, Dmitry, Andrea Vedaldi, and Victor S. Lempitsky (2017). “Deep Image Prior”. In: *CoRR* abs/1711.10925. arXiv: 1711.10925. URL: http://arxiv.org/abs/1711.10925 (cit. on p. 2).

Weigert, Martin, Uwe Schmidt, Tobias Boothe, Andreas Müller, Alexandr Dibrov, Akanksha Jain, Benjamin Wilhelm, Deborah Schmidt, Coleman Broaddus, Siân Culley, Mauricio Rocha-Martins, Fabián Segovia-Miranda, Caren Norden, Ricardo Henriques, Marino Zerial, Michele Solimena, Jochen Rink, Pavel Tomancak, Loic Royer, Florian Jug, and Eugene W. Myers (2018). “Content-Aware Image Restoration: Pushing the Limits of Fluorescence Microscopy”. In: *bioRxiv*. DOI: 10.1101/236463. eprint: https://www.biorxiv.org/content/early/2018/07/03/236463.full.pdf. URL: https://www.biorxiv.org/content/early/2018/07/03/236463 (cit. on p. 21).