EE 510: Deep Learning Theory and Practice **Winter 2020**

Course Project: Image Denoising
Due: March 15, 11:59 PM PT

Student Names: Merlin Carson, Ruchit Kank, Jean Paul Mugisha

1 Paper Descriptions

1.1 Paper 1

Paper Title: Image denoising: Can plain Neural Networks compete with BM3D? [1]
Student Name: Jean Paul Mugisha

1.1.1 Problem Description & Formulation

In this paper, the authors describe a way to denoise an image using a plain neural network. Denoising an image is a way of removing noise in an image and produce a clean image with no noise. Noise are unimportant data added to an image. These noises can be removed using different approaches that result in a clean image.

This paper presents a technique of denoising an image using a plain neural network. Using a multilayer perceptron (MLP), patches of the images are taken as input to the MLP, and they pass through different layers producing an output that is compared to a clean patch. A loss between an the MLP output and a clean patch is calculated and minimized using the backpropagation method. After training, clean patches are produced and this model can be used to denoise other images.

The technique in this paper is possible by assuming that MLP is large enough, i.e. more hidden layers. The patch size is also chosen large enough, this ensures that the patch has enough information to recover a clean version of the image. And finally, large enough training set. This ensures, that the model doesn't overfit.

Image denoising is applicable in image processing and computer vision problems. If an image is corrupted during storage or during transmission, it can result in faulty outputs depending on the application. Getting a denoising technique like the one described in the paper, ensures that issues that would result from noisy images are avoided.

This paper used two different data set. The small training set that consists of 200 images and the large training set with 150, 000 images. Below are the paramaters used in the algorithm.

- Large training set, the MLP has 4 hidden layers of size 2047, and a patch size of 17x17 (L–17–4x2047).
- Small training set, the MPL has 2 hidden layers of size 511, and a patch size of 13x13 (S–13–2x511).
- The learning rate if 0.1 is used, and no regularization on the weights.
- Stochastic gradient descent (SGD) during training.

The results of the model that's trained on the large training set are much better and this is the model that is used to compete with other denoising algorithm.

To obtain the noisy images, clean images are corrupted with Gaussian noise with sigma = 25. In the paper they performed the experiments on gray-scaled images that were obtained from color images using MATLAB's rgb2gray function.

1.1.2 Application/Model/Algorithm/Theory Description

This technique uses plain fully connected multi-layer neuron networks. The input patches are corrupted with a known additive white gaussian noise (AWG) and they are mapped onto clean patches. With back-propagation and gradient descent for loss minimization, the patches are denoised and output noise free images.

- As stated in the problem description above, for this approach to work effectively, having a large training set gives better outputs.
- Another technique used in this paper, for weight initialization, the weights are sampled from a normal distribution with mean 0 and standard deviation = sqrt(N), where N is the number of input units of the corresponding layer. (This technique was discussed in class and it was great to see it here in practice).
- The paper also uses a learning rate division. In each layer, they are dividing the learning rate by N. This helps to change the number of hidden units without modifying the learning rate.

The technique above, should work given that there is enough data and the MLP is large enough. With backpropagation, the loss function can be minimized to a minimum. Of course, this needs to be done on a GPU, since it can do matrix multiplication faster than the CPU. On a CPU it would take longer to train, that's why this algorithm takes the advantage of the GPU.

This algorithm does much better on AWG noise (Additive White Gaussian Noise), however, not all the noises are AWG. On other types of noises, the model performs poorly even when it's trained on the larger data set with more layers. When tested on AWG noise, It was able to compete with other advanced techniques and even outperformed them on some cases. For example, on the standard test images (these are 11 images used for testing different algorithm), the MLP outperforms the BM3D on 6 out of the 11 images, it outperforms KSVD on 10 out of 11 the images. BM3D and KSVD are the techniques that were state of the art at the time of the writing of the paper. More results can be found in the paper.

1.1.3 Practical and/or Theoretical Results

The results from this model are based on gray-scaled images. Its performance is measured using peak to signal noise ration (PSNR). A PSNR of a denoised image using MLP is measured and compared to PSNR of other denoising techniques. And the results depend on the number of training images used.

For the MLP model trained on the small dataset of 200 images, the PSNR after over 2 million training samples is 27, which is lower compared to other models. But with the larger dataset of 150, 000 images, the MLP model's PSNR improves to 30 and this what is used to compete with other techniques.

This model assumes that the noise is AWS. To denoise images with different noises, a different model has to be trained for that specific noise. In they paper the trained only on AWS noise and that's where all the results are based. Using this model to denoise other images, the results are poor and MLP is outperformed by other techniques like BM3D.

In summary, the problem of the denoising the image in this paper is solved with the assumption that the noise is AWG and that the model is trained on a large data set. One weakness of this approach is that it doesn't talk about the results on the colored images. We tried to replicate this model and trained it on colored images, but the model did not learn anything. One more weakness, this model has more parameters, and this takes a significant amount of time to train. In the paper, they trained the model on the large dataset for a month to get competitive results, with a CNN model that we build and trained on the same dataset, it took us a few hours to get the same results on MLP trained for a month.

1.1.4 Relation to Course Material

All the aspects in this paper are covered to the course material in the deep neural network. In fact, this is a perfect example of a multilayer neuron network in practice. It requires a little bit of data processing of image pixels in order to get patches with needed size, but other than that implementing it using Pytorch is straight forward.

Some of the topics covered in class that were used in this paper include weight initialization, SGD, and most importantly, this paper emphasized the importance of training on more data and having enough layers. The paper also discusses about how they didn't use many hidden layers in order to avoid vanishing gradient and over-fittings problems, these are the topics that we discussed in class too.

I could not have understood any portion of the paper if I hadn't taken EE510 (I don't have a CS background, but I wanted to get an introduction on Deep Learning and AI in general). When I walked in my first lecture of this class, I felt lost, I didn't know whether to continue or drop the class, but with more practice and the weekly reading, I started to understand the class material, and I was able to understand this paper.

1.2 Paper 2

Paper Title: Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising [2]
Student Name: Ruchit Kank

1.2.1 Problem Description & Formulation

The authors present that in the field of image denoising, the focus is on tackling additive white Gaussian noise (AWGN) v of the form $y = x + v$, where x is the clean image and y the noisy image. They consider discriminative model learning as a promising candidate for their application. Their proposal describes using a CNN to learn noise from an input injected with random and specific noise levels i.e. learn the mapping $R(y) \approx v$

Image denoising is a critical component in image preprocessing and allows algorithms more significant data to work with. Gaussian noise is one of the most prevalent forms of noise in digital images as an artifact of camera sensors and it is additive and independent at every pixel. Consequently, a technique for good, fast and computationally inexpensive AWGN denoising can boost availability of clean image data for research or in real-time computer vision applications.

While prior works in discriminative models for denoising have shown an improvement over conventional methods in computational efficiency, their reliance on the image priors made them unsuited to generalizing denoising tasks beyond the specific types of priors encountered in training and noise level used. Further, the models also involved manual parameter tuning. The authors address these shortcomings by using CNNs to predict the noise mapping. By removing the clean image from the layers, their model can learn to be prior invariant and predict strictly on the noise characteristics common to them; this property can also be extended to applications beyond Gaussian denoising. CNNs allow flexibility in the input, have considerable optimization techniques available and are a great candidate for parallel computation. Their results show that their blind DnCNN performs better than state-of-the-art methods tuned to specific noise levels and the model can generalize to tasks such as SISR and JPEG deblocking on the same model.

1.2.2 Application/Model/Algorithm/Theory Description

Their architecture is a CNN of D layers: the first layer is a Conv+ReLU that uses 64 3x3xc filters, (D-2) layers of Conv+BatchNorm+ReLU with 64 3x3x64 filters, and a Conv layer with c 3x3x64 filters, where c is the number of input image channels. The model learn to predict the noise with a MSE loss

$$l(\Theta) = \frac{1}{2N} \sum_{i=1}^{N} ||R(\mathbf{y}_i; \Theta) - (\mathbf{y}_i - \mathbf{x}_i)||_F^2 \tag{1}$$

where Θ are parameters learned

By learning the Gaussian noise and omitting the image, the model can identify the common AWGN features on a wide array of images without the priors affecting the learning in a significant way.

Residual mapping optimizes easier if the mapping from predicting the clean image is linear, which holds true in the case of DnCNN. They observed faster convergence with residual learning on both the SGD and Adam optimizers. While batch normalizing was used to speed up training, the model also discovers that residual learning and batch normalization synergize well; residual learning preserves only the Gaussian noise characteristics in its layers that batch normalization can exploit to reduce the internal covariate shift.

They obtain considerable improvements over BM3D in not only PSNR but also the quality of the image in sharpness of edges and color details. The model performs poorly only on images with repetitive structures as learning noise strips away the image data that makes non-local methods work better.

1.2.3 Practical and/or Theoretical Results

Two variants of the model were trained: DnCNN-S trained on patches sized of 40 x 40 with known noise levels, $\sigma = 15, 25, 50$, and DnCNN-B trained on patches sized 50 x 50 with blind Gaussian noise levels $\sigma \in [0, 55]$. The CDnCNN-B model with color images used 432 images from the Berkeley segmentation dataset and was tested on the BSD68 dataset. DnCNN-S trained on with 17 layers with a SGD optimizer, weight decay of 1e-4, momentum of 0.9, mini-batch size of 128 and exponential learning rate of 1e-1 to 1e-4 over 50 epochs.

They achieved better average PSNR and structural similarity over benchmark BM3D as well as WNMNM, EPLL, MLP, CSF and TNRD models for varying noise levels using a single model. GPU inference time also beat that for other models like TNRD while producing a better denoised image. The same model trained for single image super resolution and JPEG image deblocking called DnCNN-3 was also found to match or outperform state-of-the-art models in those tasks.

However, their model is only tested on synthetic benchmarks against other models. This can be attributed to the synthetic benchmarks sufficiently simulating real noise and tasks like super resolution cannot have a real dataset. A handful of natural image noise datasets have been created only recently and lack the amount of data models working with images can need.

1.2.4 Relation to Course Material

The paper didn't propose any architecture or optimization unfamiliar to me. CNNs were an important part of the course and the assignment on CNNs was useful in getting started with this model. ReLU and batch normalization were also discussed in detail. Loss used a simple MSE function albeit on a different target prediction. Both the SGD and Adam optimizers have been used throughout coursework and concepts of mini-batches, weight decay and momentum were explained and implemented. The only concepts that were untested was variable learning rates that we did learn of, but did not apply, and PSNR which while isn't complex, it is something I was unfamiliar with.

Having had no practical machine learning experience prior to this class, I would have likely not grasped anything beyond what the paper tried to achieve in learning the noise instead of the image. At the end of the class now, I am quite confident in being able to understand and implement the model as described in the paper

1.3 Paper 3

Paper Title: Deep Residual Learning for Image Recognition [3]
Student Name: Merlin Carson

1.3.1 Problem Description & Formulation

In 2015 researchers at Microsoft conducted tests on how add layers to a deep neural network effects error rate. They found that simply adding stacking layers past about 20 deep resulted in a higher error rate. They found this to not be an overfitting problem, since both the validation and training error was higher. Instead, they determined this to be an optimization problem. As an error is backpropagated back through significantly deep networks, the gradient gets smaller and smaller due to the multiplicative property of the chain-rule.

1.3.2 Application/Model/Algorithm/Theory Description

To address this problem they developed a deep residual learning framework. Instead of hoping each few stacked layers would learn an underlying function, they built what they called a residual learning block, allowing stacked layers to learn a residual mapping.

$$F(x) := H(x) - x \tag{2}$$

where $F(x)$ represents the stack of layers within the residual block, $H(x)$ represents the residual mapping and x is the input signal to the block

This mapping is accomplished by what the authors dubbed "shortcut connections", now more commonly known as skip connections. These shortcuts would route the incoming signal to the block around the layers and the summation of the signal passed through the layers and this incoming signal, known as the identity, would be pass through a non-linearity before being passed to the next residual block. This reformulated the problem as,

$$H(x) := F(x) + x \tag{3}$$

where $F(x)$ is output of the layers in the residual block given the input signal ans x is the identity signal for the block.

Adding these skip connections not only allow the network to be trained end-to-end, but also do not add any parameters or computational complexity.

1.3.3 Practical and/or Theoretical Results

With these new residual learning blocks, the authors were able to successfully train a 152-layer deep convolutional network and achieve a 3.57% top-5 error on ImageNet with an ensemble of these models, thus winning the competition. The models contain 8 times the number of layers as the previous year's winner, VGG-19, with a low complexity. The models not only won the ImageNet detection competition in 2015, but also the ImageNet localization, COCO detections, and COCO segmentation. They also demonstrated similar increases in performance on the CIFAR-10 dataset, showing that residual learning properties are not dataset specific, but are applicable to a variety of other vision and non-vision problems.

1.3.4 Relation to Course Material

While much of the math and concepts in this paper are generic and can be applied to multi-layer-perceptron models, this paper specifically focuses on convolutional neural networks. Both the VGG-19 and ResNet-152 are discussed in the paper as were in class. When discussing the layers in these networks, filter size, stride, and padding are all specified. The paper talks about using 1x1 convolutions as well as pooling operations for downsampling within the network. The paper also goes into details related to model complexity, number of parameters and FLOPs that the models require to run. It discusses regularization techniques such as weight decay, drop out and batch normalization.

Data handling is of course a major topic in any deep learning class. The paper discusses the size of the training, validation and test splits of the underlying dataset. It also discusses specific types of data

augmentation. Normalization for the data is also detailed. They also discuss how the complexity of the model and the size of the dataset can cause overfitting.

Optimization is a large part of the paper, since their intuition about why stacking a significant amount of layers increase the error rate is an optimization problem. The paper describes the stochastic gradient descent parameters, weight decay, momentum and batch size. It also goes into the learning rate scheduler, what the initial learning rate is and how it is annealed over time. And it discusses the number of iterations it for their model to converge. All of these topics were discussed in our class.

2 Comparison of Approaches/Models/Algorithms/Theories

- Interpretability: For the most part the papers were easy to follow, given the knowledge gleaned from this class. The MLP and the DnCNN models are fairly straightforward architectures. Therefore, using PyTorch makes neither the MLP nor the CNN particularly challenging to implement. The CNN model certainly stands out as the superior model in not being constrained to an input size and performs better on image tasks. As observed during training, the CNNs quickly achieve a high PSNR that the MLP did not match even after 50 epochs.

  The DnCNN paper, being a much more modern paper, discussed most aspects of the model's hyper-parameters and training regime fairly well. It also uses standardized nomenclature that has become common as Deep Learning has become a more mature field in the last few years.

  The MLP paper left out many key points in how the model was tuned and trained. It used many ambiguous terms like iterations and how many examples the model was presented between testing its performance. Terms like epochs and size of training set were not discussed.

  The ResNet paper is a little less interpretable. The general implementation details of the residual blocks and the ResNet models are well defined. However, the reasoning behind why residual blocks increase the performance of ultra deep models is some what nebulous. The paper also brushes over why optimization fails when trying to stack layers into ultra deep models. They could not find the cause for increased error rates in significantly deeper models but eliminate several suspects.

- Theoretical guarantees: Neither of the papers made many theoretical guarantees. However, it is made clear from the DnCNN paper that CNN models perform better than MLPs on vision related tasks due to them maintaining some spatial locality as the examples are fed through the model. The DnCNN paper also discusses the theoretical advantage that batch normalization adds to the process of the residual learning of noise from a Gaussian distribution, due to their reduction in internal covariate shift.

  The ResNet paper does discuss some theoretical guarantees, not in the exact performance, but that adding skip connections allow the model to learn residual functions. And that the blocks during training will learn to zero out the weights in the blocks and rely on the signal from the skip connection if the residual function closely mirrors the identity.

- Empirical guarantees: Both papers show reasonable results on black and white images. However, the DnCNN model performs better than the MLP model on all of the standard images common to both papers. The DnCNN can also learn multiple noise levels on which the MLP struggles to give good results. In addition, the DnCNN paper shows impressive results on color images too. The DnCNN is also capable of handling images of any height and width due to the sharing of weights in the convolutional layers, so its much easier to process and denoise images of any size. The MLP requires a complex procedure of stitching and smoothing deconstructed images to achieve its best results.

  The ResNet paper provides very specific empirical improvement guarantees given their new architecture. These guarantees are backed up with results of their model on several datasets. The conclusion of the paper is not only that CNNs are better for vision related tasks, but that deep models with skip connections increase the performance on these types of tasks. The fact that the model won both

ImageNet and COCO competitions showed how sound their results were. This makes their residual blocks a perfect complement to the architecture of the DnCNN to increase its performance.

- Computational complexity and training/evaluation time: The MLP is overwhelmingly the most computationally complex model with 14 million parameters and training time of around 90 minutes per epoch for a fixed input size of 17x17x1. The number of parameters will grow rapidly for larger inputs and the model will need to be retrained for that input size.

  The CNN provides better and faster performance independent of the input size, clocking in around 3 minutes per epoch during training, given the same size of dataset. It only needs to be retrained to switch the number of input channels (color vs black and white). The CNN is undoubtedly the less complex, more efficient and flexible architecture for this task.

  The ResNet-152 from the paper is the successor to the VGG-19 in terms of successful image classification architecture. With their new residual blocks they were able to make a significantly deeper model, 8x the depth of VGG-19, but with much lower complexity. The ResNet-152 had fewer parameters than VGG-19 which resulted in 11.3 FLOPs vs the VGG-19's 19.6 FLOPs, a nearly 43% reduction in computation.

3 Implementation & Testing

3.1 Pre-Processing

Since the algorithms we chose involve image denoising, the first step of implementation was creating training data. Both algorithms involved training on overlapping patches from images. One of the hyperparameters for each of the models was the size of these patches, so this had to be a configurable parameter to our data generation script. fig. 1 shows sliding windows that create 50x50 training patches with a stride of 10.

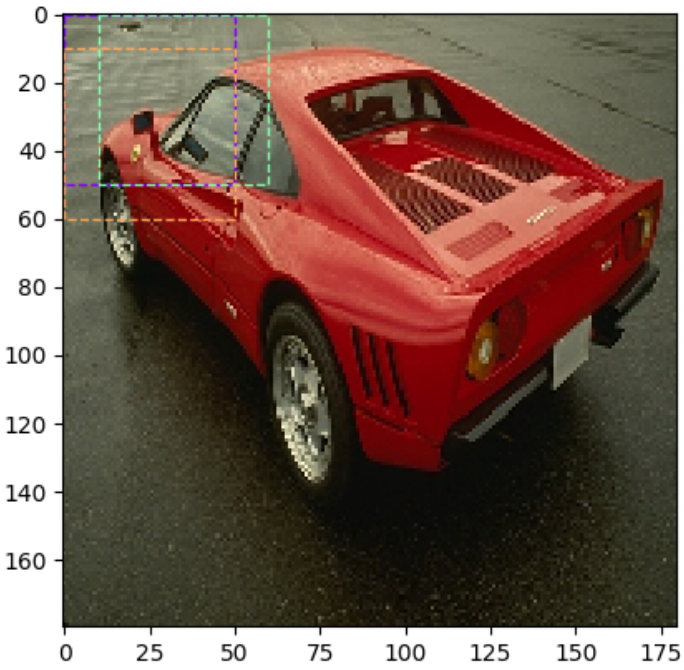


Figure 1: Training data created from sliding an overlapping window across each image in the dataset to create patches.

3.2 Data Augmentation

Training data is hard to come by, yet deep neural networks require a lot to attain a low error rate without overfitting. To extend the size of our training set, several types of data augmentation are applied to each of the images in the dataset before the overlapping patches are taken. Each image is scaled to 4 different sizes, 100%, 90%, 80%, 70%, as can be seen in fig. 3.

(a) 100%

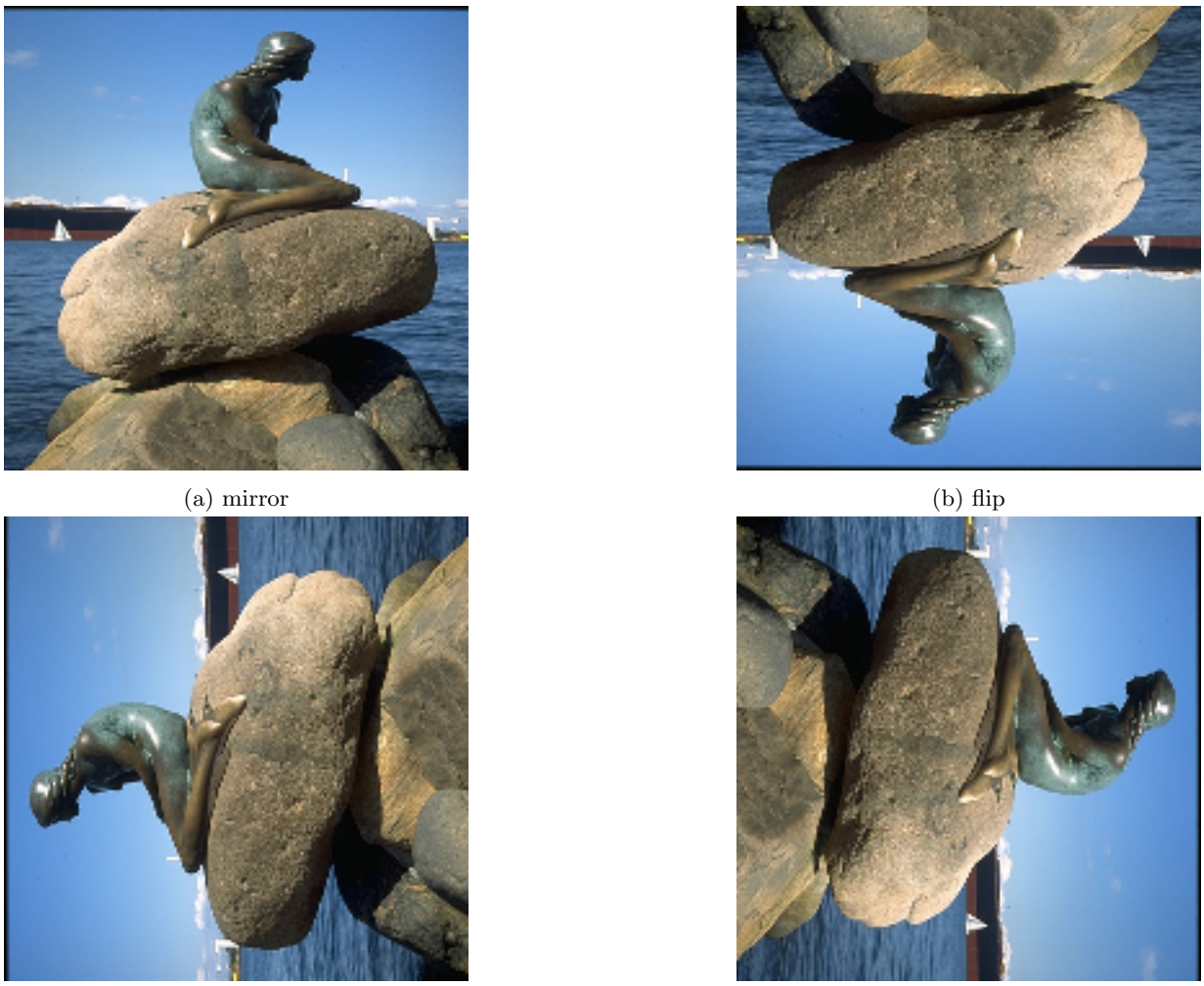
(b) 90%

(c) 80%

(d) 70%

Figure 2: Multiple scaling factors were applied to each image in the dataset to augment the training data.

For each image at each scale, mirror, flip and rotation is applied, as seen in fig. 3.



(a) mirror

(b) flip

(c) rotate left

(d) rotate right

Figure 3: Transformations to images in dataset to augment training data.

3.3 Models

3.3.1 MLP (Multilayer Perceptron)

An MLP is a network composed of multiple layers of perceptrons that map a vector input to a vector output via hidden layers. In this project we took the image patches with added noise as input and after passing through the hidden layers, an output is produced that is compared to a noise free patch. Below are the parameters that we used for our network.

- Patch size (input) = 17x17
- Number of hidden layers = 4
- Size of hidden layer: = 2047
- Output size = 17x17
- ReLU activation

- The model has 14 million trainable parameters

The training examples consist of 17x17 overlapping patches from images in the dataset. For the input to MLP model we had to flatten each example to 289x1. The output produced is of size 289x1. It had to be reshaped back to the original shape (17x17) before loss calculation. For more on how the model was trained, please refer to the training section.

3.3.2 DnCNN-B

The DnCNN-B model uses a VGGNet model modified for denoising and trained for residual learning on Gaussian noise in the input instead of predicting the clean image. Reduction in internal covariate shift from batch normalization helps the residual learning perform better. Simultaneously, residual learning keeps only the Gaussian-like noise data in the layers to help batch normalization. The definition for the model can be seen below.

- 20-layer deep fully convolutional network
- Architecture:
  - Input layer is a 2D-Convolutional layer with ReLU activation
  - 18 DnCNN-Blocks
    - 2D Convolutional layer
    - Batch Normalization
    - ReLU activation
  - Output layer is a 2D-Convolutional layer with linear output
- All convolutional layers have 64 3x3 filters with a stride of 1
- All convolutional layers have a padding of one to maintain the width and height of input through the model
- The model has 669 thousand trainable parameters

3.3.3 DnCNNRes-B

The DnCNNRes-B is a slight modification to the DnCNN-B architecture. For each pair of the repeating Convolutional + Batch Normalization + ReLU layers, between the input and output of the pair a skip connection is added to create residual blocks. This connection adds the input to the block to the output of the layers in the block before passing it through a non-linearity to generate the output of the block. This allows us to successfully train a significantly deeper, fully convolutional network, similar to the original DnCNN-B architecture. The definition for the model can be seen below.

- 82-layer deep fully convolutional network
- Architecture:
  - Input layer is a 2D-Convolutional layer with ReLU activation
  - 40 DnCNNRes-Blocks
    - 2D Convolutional layer
    - Batch Normalization
    - ReLU activation
    - 2D Convolutional layer
    - Batch Normalization

    - ∗ ReLU activation of the sum of the input to the block and the output of the layers within the block
  - Output layer is a 2D-Convolutional layer with linear output

- All convolutional layers have 64 3x3 filters with a stride of 1
- All convolutional layers have a padding of one to maintain the width and height of input through the model
- The model has 2.69 million trainable parameters

3.4 Training

All three models were trained in a similar fashion. The clean training patches were iterated over in batches. The models were trained on noise from a normal distribution (Gaussian noise), uniform distribution (bright noise), and on pepper noise (pixels blacked out). Varying levels of noise is uniformly applied across the batch. For Gaussian noise, the first example in the batch has the lowest noise level, $\sigma = 0$ and the last example in the batch has the highest noise level, $\sigma = 55$. For uniform and pepper noise, the first example has the lowest noise level, 0% of the pixels are corrupted, and the last training example has the highest noise level, 25% of pixels are corrupted. Examples of this pre-processing can be seen in fig. 4. Training on varying levels of noise is called blind training, hence the model names containing "-B" after their names.

(a) Gaussian noise

(b) Uniform noise

(c) Pepper noise

Figure 4: Different types of noises used to train the models.

The DnCNN models are trained using Mean Squared Error (MSE) with the error calculated sample-wise, not pixel-wise, between the predicted noise and the generated noise, thus, optimizing the model to learn the residual mapping between a noisy image patch and the noise in the patch. The MLP model uses the same loss function, but calculates the error between the predictions and the clean image patch. Thus learning a direct mapping between a noisy image and a denoised image.

The gradient of a batch size of 128 is collected between each weight update. The adaptive moment estimation (Adam) optimizer [4] is used for backpropagation. Adam parameterizes its weight updates by tracking both the average of the gradients and the second moments of the gradients. An initial learning rate of $\eta = 1e{-}2$ is applied to the optimizer.

Exponential learning rate decay scheduling with $\gamma = 0.87$ is executed during training. This anneals the learning from the initial $\eta = 1e{-}2$ to $\eta = 1e{-}5$ for a total training duration of 50 epochs. This can be seen in fig. 5.

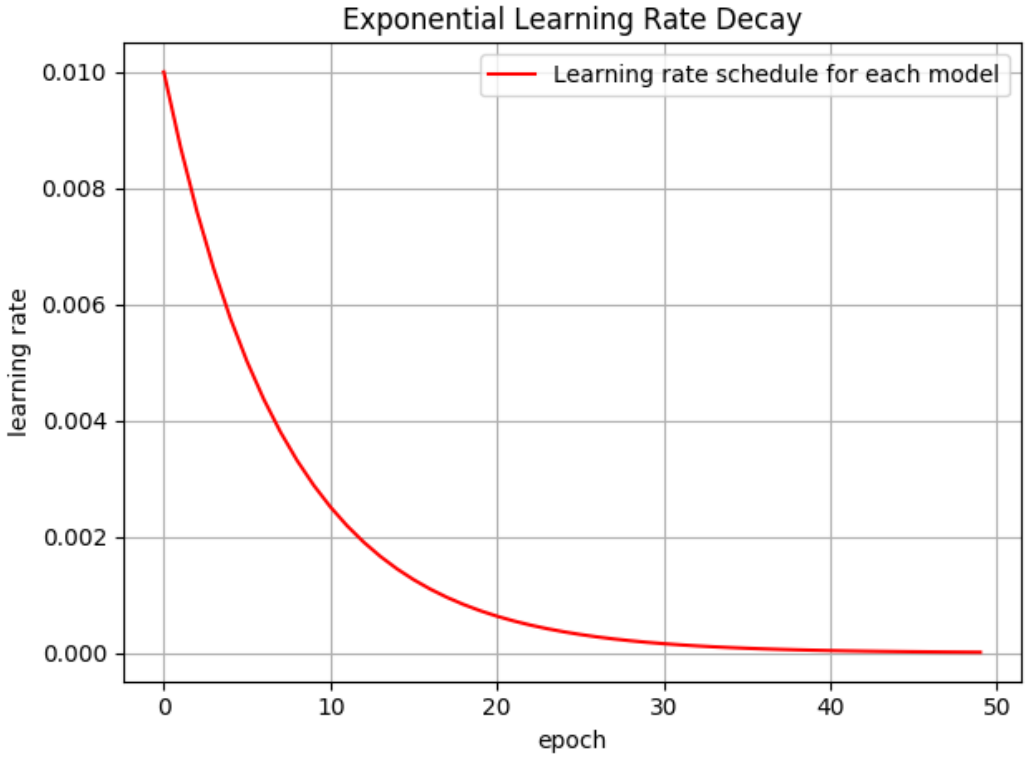


Figure 5: Exponential learning rate decay used to train models.

Between each batch of training data, the model is validated on a held out portion of the original dataset. The validation set is also comprised of patches, but with no data augmentation. Noise is uniformly added across the batch the same way it is applied during the training phase. The MSE loss, along with the average PSNR on the set is calculated to determine when overfitting begins and therefore when to save the best model.

4 Experiments

4.1 Multi-Layer-Perceptron

Our first experiment with the Multi-Layer-Perceptron (MLP) [1] was using the Berkeley Segmentation dataset BSDS300. The MLP paper uses 400 black and white images from this dataset, segmented into 17x17 overlapping patches. With data augmentation, this generated 1.6 million training examples, which totals 4.1 GiB of data. With the model's significant amount of trainable parameters, over 14 million, it took approximately 50 hours to train on 3 Nvidia 2080 Ti GPUs. The training curves in fig. 6 show a steady decrease in both training and validation loss. Overfitting did not occur and the model could probably have continued training for a longer amount of time, but the paper did not specify how many epochs the model was trained for. However, as can be seen in fig. 7, while the PSNR on the validation set was still increasing, its improvement was seeming to level off. It's doubtful that there would have been a noticeable increase in PSNR without a significant increase in the training duration.

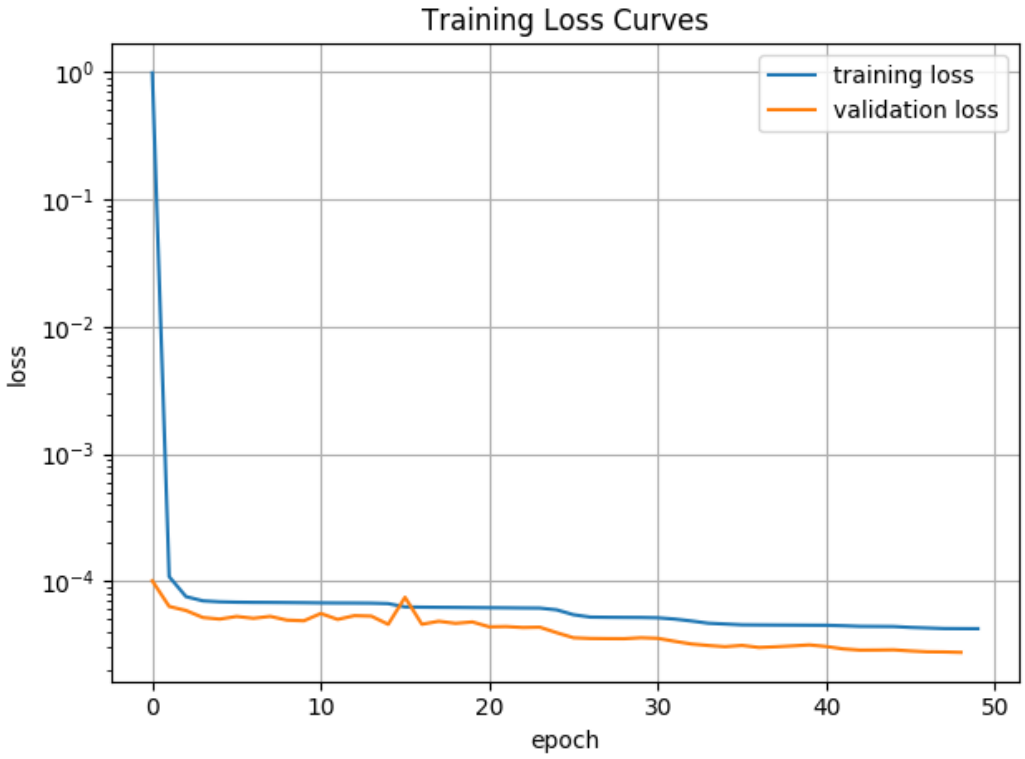


Figure 6: Loss curves for MLP black and white Gaussian noise experiment.

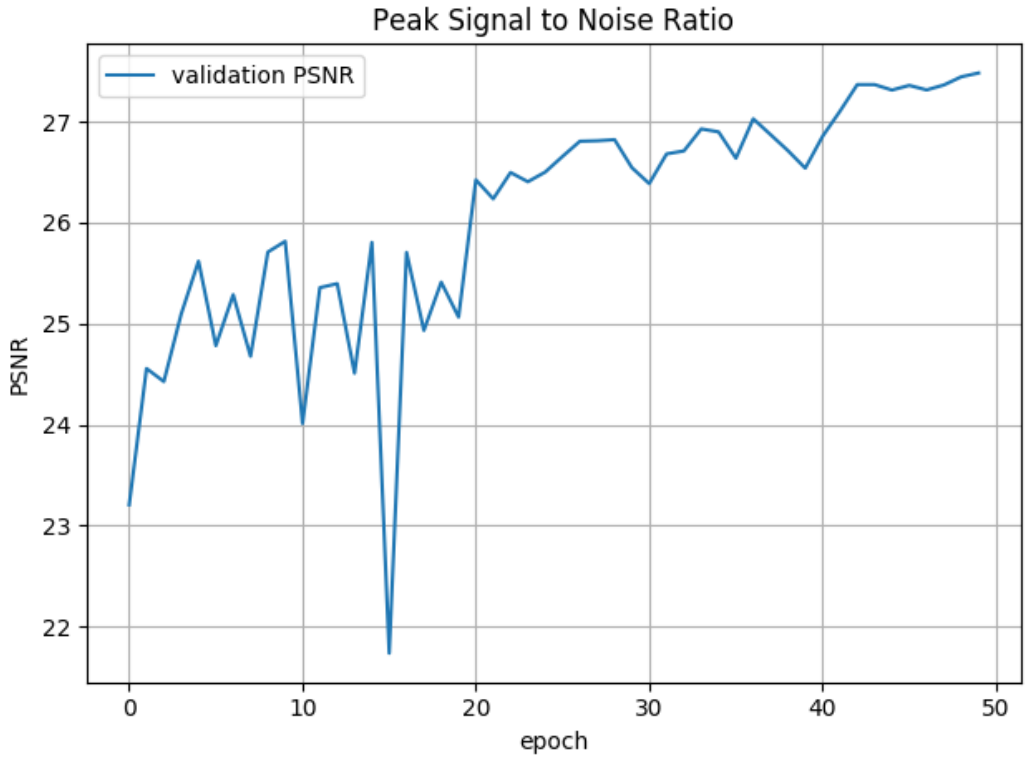


Figure 7: PSNR for validation set during MLP black and white Gaussian noise experiment.

Despite the MLP paper focusing on denoising black and white images, our focus was to compare the results of our models to state of the art denoising algorithms. Since the other algorithms performed well on color images, we attempted to train the model on color images. For this we used the Berkeley Segmentation dataset BSDS500, which contains 400 full color images. The network was modified to account for the 3 color channels, so the 17x17x3 patches flattened now required an input and output size to be 867. This blew the number of trainable parameters up to over 16 million. Ultimately the model failed to learn to reconstruct the denoised images properly and never achieved over a 12.45 PSNR, lower than the noisy pictures used for the input.

It's not surprising that the MLP failed to learn to reconstruct color denoised images. Unlike a convolutional neural network, the spatial information of an input image is not maintained once fed into an MLP network. While the black and white version of the MLP was eventually able to learn how to reconstruct the spatial information after a few epochs, the color version would have to learn how to appropriately layer this information for the proper color channels. It's possible that given a sufficiently larger network, the model might have been able to learn this behavior, but the model would probably required 3x the number of parameters. This would have required significantly reducing the batch size of the model to fit into GPU memory, which along with a much larger set of parameters would have taken in intractable amount of time to train.

Another drawback to MLP model is that there is no weight sharing like in a convolutional model. The input and output to the model is fixed and unidimensional. So, visualizing the results of the denoising process would require decomposing an image into patches, processing them through the model, then stitching the results back together at the end. The paper describes a smoothing process along the boarders to achieve the best results. Because of this extra complexity, we have no visual results from the model.

4.2 DnCNN-B

The second model we ran experiments on was the DnCNN-B [2]. Initial tests were performed on our implementation to training on the BSDS300 black and white images. Once it was shown that the model functioned well, we switched to training on full color images from the BSDS500 dataset. 380 images were used for training, while 10 images were set aside for validation and 10 images for testing. Having overlapping patches of 50x50x3 with a stride of 10, along with applying all previously mentioned data augmentation, 796,000 training examples were created, totally 56 GiB of data. However, despite a significantly larger dataset, the model trained much faster than the MLP, approximately 16.6 hours on 3 Nvidia 2080 Ti GPUs. This is due to a significantly smaller number of trainable parameters, 669 thousand vs the MLP's 14 million. The number of training examples is also less because of the larger patch size, thus resulting in fewer weight updates per epoch.

Despite an initial spike in the validation loss a few epochs into training, the loss curves were fairly smooth, as can be seen in fig. 8. No signs of overfitting were seen during the 50 epochs of training, however it can be seen in fig. 9 that the PSNR improvement began to level off around 30 epochs in. The validation PSNR mirrored the results in the DnCNN paper fairly well. But when testing the best model on the same benchmark dataset used in the paper, our model performed with a nearly 15% better PSNR improvement. The paper does not explicitly state which dataset was used for training, just the number of images and size of patches. So our results are probably related to the dataset that we chose. It's also possible that improvements to the learning algorithms in PyTorch could account for some of the differences between our results and the 2017 paper.

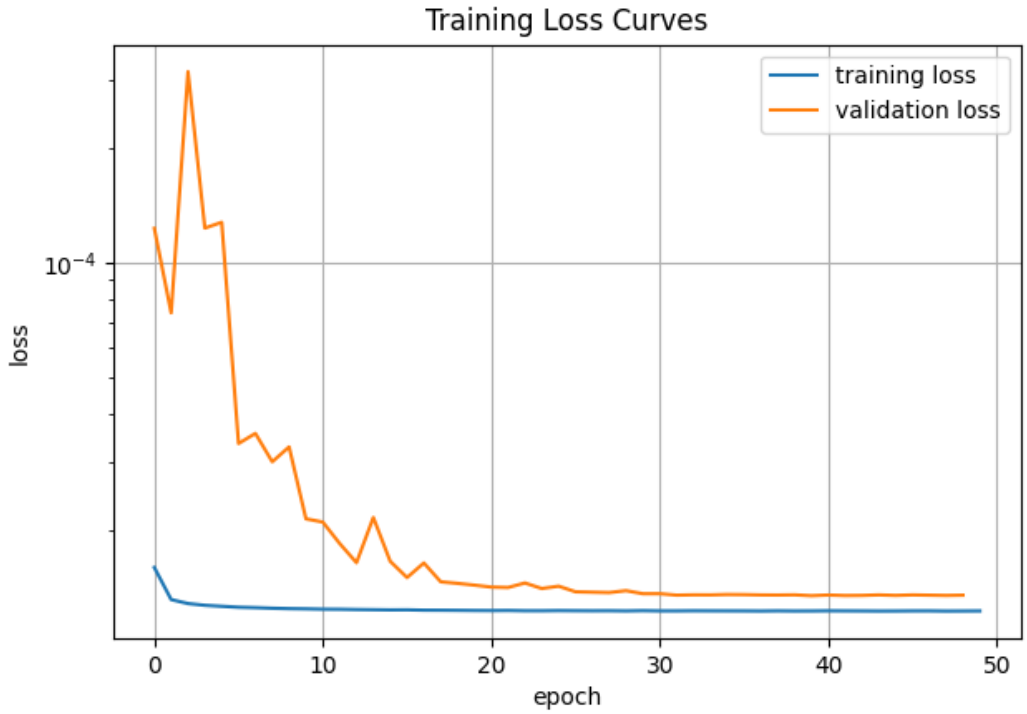


Figure 8: Loss curves for the DnCNN-B Gaussian noise experiment.

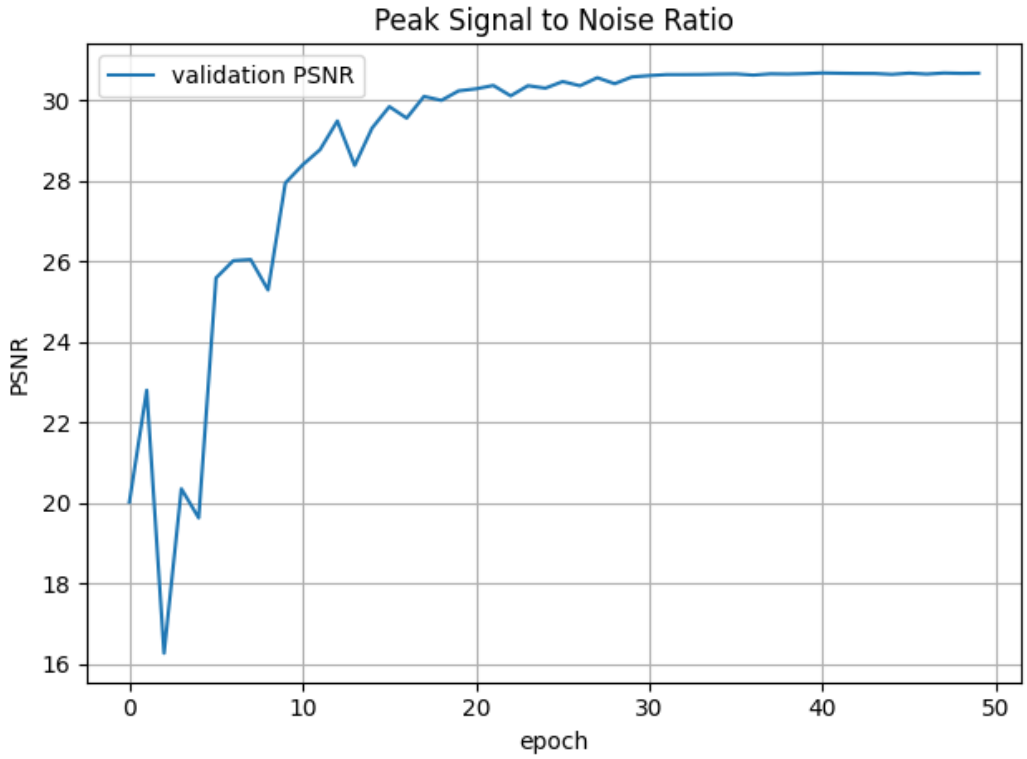


Figure 9: PSNR for validation set during DnCNN-B Gaussian noise experiment.

Results of the model with different amounts of noise on an image with distant features can be seen in fig. 10. Results of the model with different amounts of noise on an image with near features can be seen in fig. 11. The model does fairly well at all noise levels. However, at higher noise level there is a higher level of smoothing that occurs. The lower the noise level, the more distinct fine grained features are maintained.

(a) Noisy image $\sigma = 50$

(b) Denoised image

(c) Noisy image $\sigma = 25$

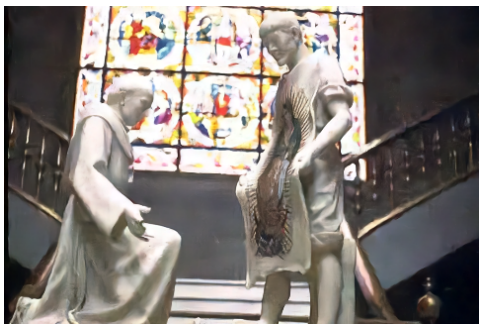
(d) Denoised image

(e) Noisy image $\sigma = 15$

(f) Denoised image

Figure 10: Example of DnCNN-B denoising on images with different noise levels.

(a) Noisy image $\sigma = 50$



(b) Denoised image

(c) Noisy image $\sigma = 25$

(d) Denoised image

(e) Noisy image $\sigma = 15$

(f) Denoised image

Figure 11: Example of DnCNN-B denoising on images with different noise levels.

4.3 DnCNNRes-B

The DnCNNRes-B model showed almost immediate better results in PSNRs compared to it's little brother the DnCNN-B, as can be seen between fig. 13 and fig. 9. However, significantly more variance can be seen in the validation curve fig. 12 and the PSNR curve fig. 13 compared to the DnCNN-B model. The validation curve especially showed vary erratic behavior past 15 epochs, that we cannot explain, but it did not immediately correlate to overfitting, since the validation PSNR continued to increase. Unlike the smaller DnCNN-B model, which did not show signs of overfitting through the entire 50 epoch training, the DnCNNRes-B started to show degradation in the PSNR values after about 37 epochs. This is not surprising due the extreme complexity of the model. Given a larger training set and training time, the model should be able to train longer without overfitting and achieve an even greater PSNR improvement.

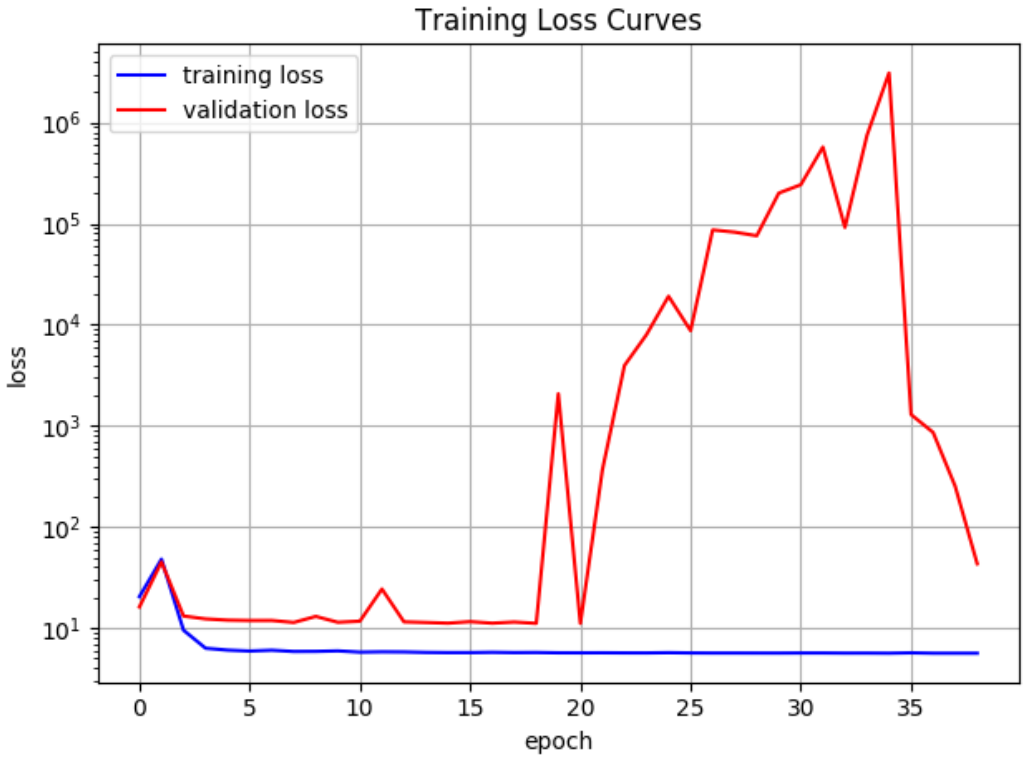


Figure 12: Loss curves for the DnCNNRes-B Gaussian noise experiment.

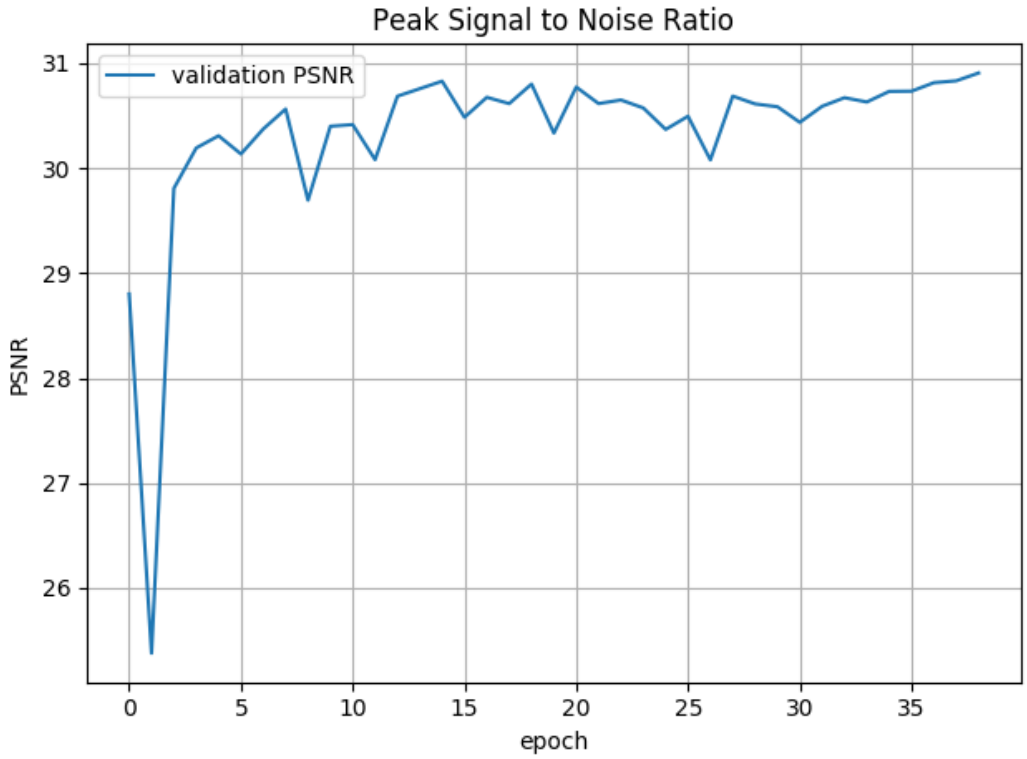


Figure 13: PSNR for validation set during DnCNNRes-B Gaussian noise experiment.

fig. 14 show the a comparison between the DnCNN-B and DnCNNRes-B models on an image with distant features at different noise levels. fig. 15 show the a comparison between the DnCNN-B and DnCNNRes-B models on an image with near features at different noise levels. A quick look might not reveal much difference between the two. However, upon closer inspection, the DnCNNRes-B model produces images more representative of the features of the original image with fewer artifacts and lower loss of detail.

(a) DnCNNRes-B denoised, $\sigma = 50$

(b) DnCNN-B denoised, $\sigma = 50$

(c) DnCNNRes-B denoised, $\sigma = 25$

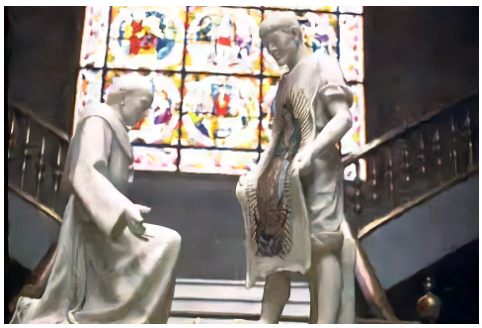
(d) DnCNN-B denoised, $\sigma = 25$

(e) DnCNNRes-B denoised, $\sigma = 15$

(f) DnCNN-B denoised, $\sigma = 15$

Figure 14: Comparison of the DnCNN-B and DnCNNRes-B image denoising at different noise levels.

(a) DnCNNRes-B denoised, $\sigma = 50$



(b) DnCNN-B denoised, $\sigma = 50$

(c) DnCNNRes-B denoised, $\sigma = 25$

(d) DnCNN-B denoised, $\sigma = 25$

(e) DnCNNRes-B denoised, $\sigma = 15$

(f) DnCNN-B denoised, $\sigma = 15$

Figure 15: Comparison of the DnCNN-B and DnCNNRes-B image denoising at different noise levels.

Seeing the best results on the DnCNNRes-B model, we chose to train the model on different noise types other then the Gaussian noise discussed in the denoising papers. Other synthetic noise that models real image degradation is uniform and pepper noise, as discussed above.

The best results of all came from training the model on pepper noise. As can be seen from the loss curves, fig. 16, overfitting did occur within the 50 epochs. There also were not large spikes in the validation curve, as seen when training the model with Gaussian noise. While there was a fair amount of variance in the PSNR curve, as can be seen in fig. 17, the overall trend was increasing over time. One important note is that pepper noise blacks out pixels, which has a much more corrupting effect on an image's PSNR. The highest Gaussian noise level example the model was trained on had a PSNR of 13.34, while the highest pepper noise level example had a PSNR of 6.03. So, when the model does a good job of denoising an image with pepper noise, it achieves a significantly larger improvement in PSNR. Therefore, as can be seen in the PSNR graph, the model shows much larger values then that of the model trained on Gaussian noise. However, it can still be noted that the model did significantly better learning to denoise pepper noise vs. Gaussian noise. This is not surprising since Gaussian noise adds small amounts of noise, since it's centered at 0, to all the pixels

in an image. Pepper noise is simulating missing data or debris on a lens instead, so some percentage of pixels are affected and the rest are unaffected. This results in extreme gradients in the images around the corrupted data, which the model should be able to identify very well.

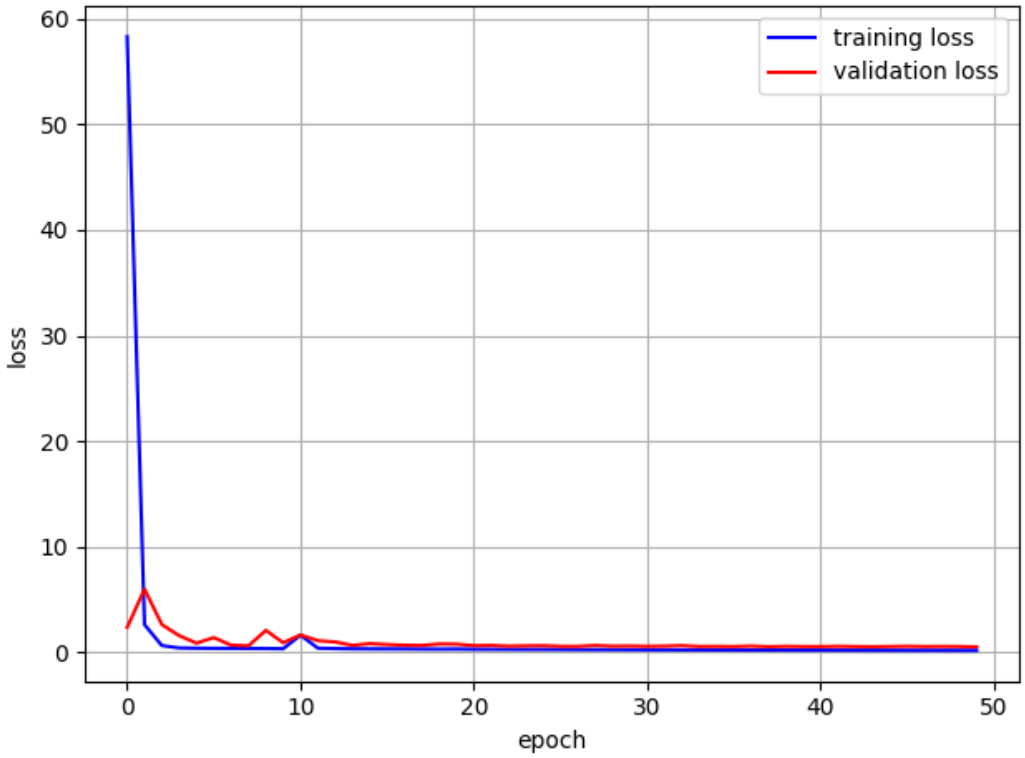


Figure 16: Loss curves for the DnCNNRes-B pepper noise experiment.

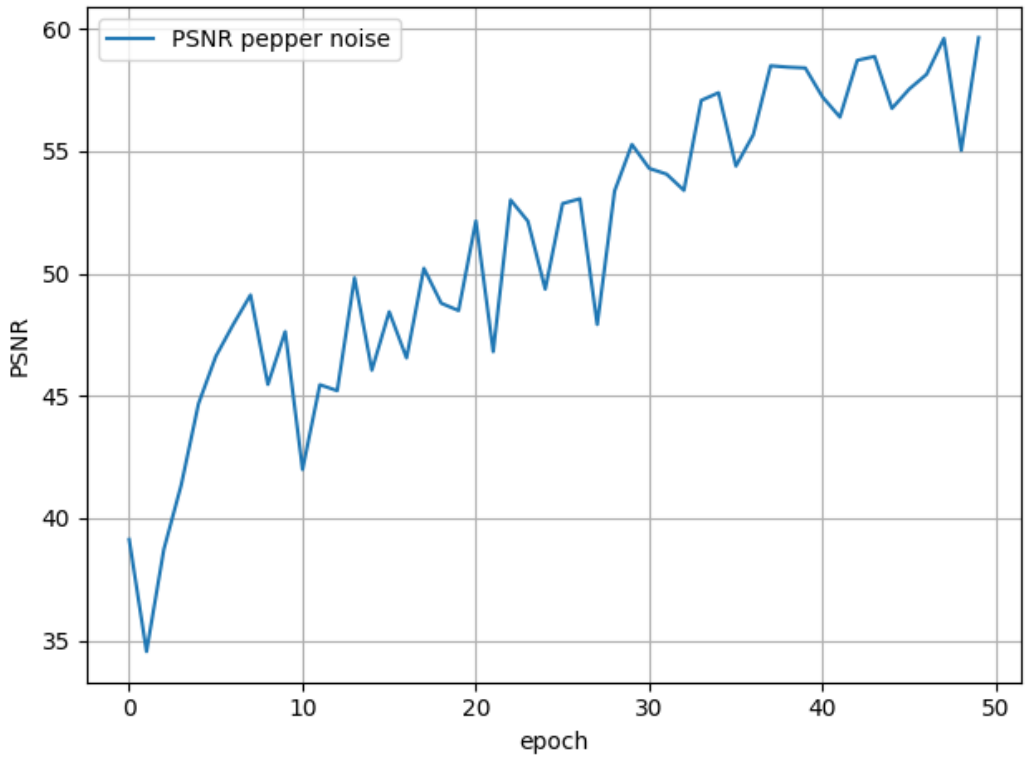


Figure 17: PSNR for validation set during DnCNN-B pepper noise experiment.

Models trained on uniform noise and on all 3 noise types did not perform as well, as can be seen in fig. 18. Only 8 epochs are shown in the graph since the PSNRs oscillated between the shown values and then began to degrade over the duration of training. The curve for the Gaussian noise is significantly lower than the models trained on this noise alone and did not show any obvious signs of improvement. The PSNRs on uniform and pepper noise had higher overall values, but the PSNR range for these more corrupting types of noise is not high enough to generate images that are perceptible noise free.

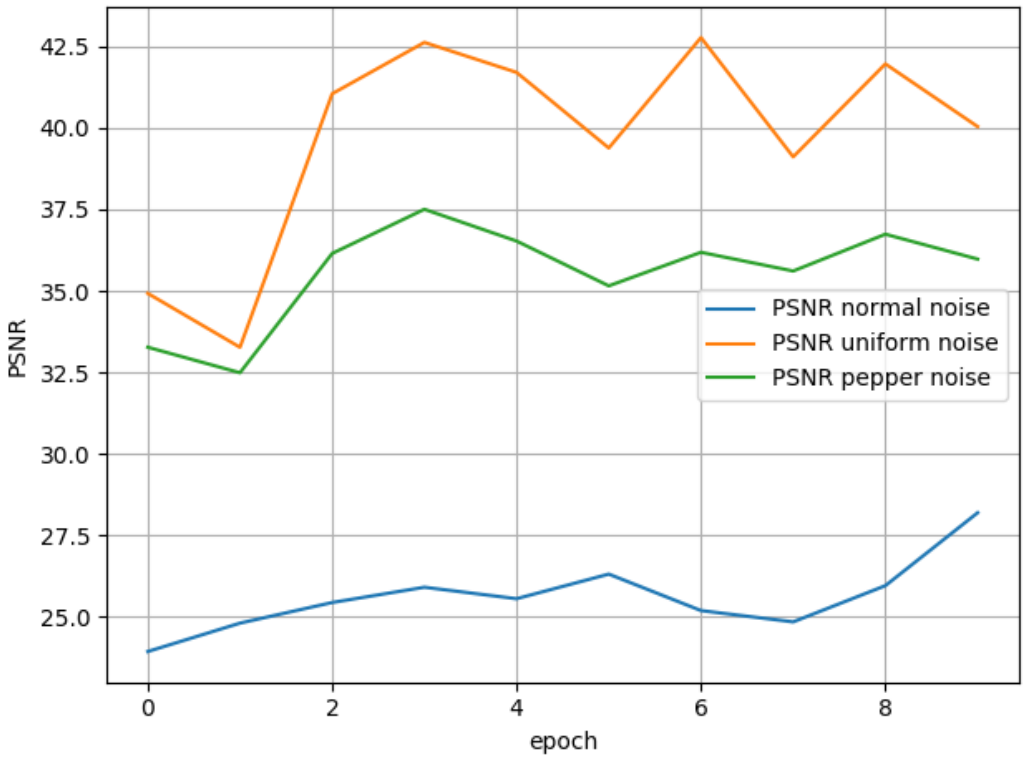


Figure 18: PSNRs for validation set when training DnCNNRes-B on 3 noise types.

4.4 Results on Benchmark Dataset

The DnCNN-B and DnCNNRes-B were evaluated using the Berkeley Segmentation dataset CBSD68. This dataset consists of 68 full color images and is the same benchmark dataset used in the DnCNN paper to compare its results to other state of the art denoising algorithms. The comparison between the results in the paper on this dataset and the results of our models can be seen in table 1. Our implementation of the DnCNN-B achieved slightly better results than the paper cites on the same dataset, especially at higher noise levels. However, the DnCNNRes-B achieved almost a 22% PSNR improvement after denoising than the original DnCNN-B did on the dataset. The average PSNR improvements can be seen in table 2.

Table 1: Results of state of the art denoising algorithms on the CBSD68 benchmark dataset

Noise Level	BM3D	TRND	DnCNN-B (paper)	DnCNN-B (ours)	DnCNNRes-B
$\sigma = 15$	31.07	31.42	31.61	31.79	33.91
$\sigma = 25$	28.57	28.92	29.16	31.13	31.27
$\sigma = 50$	25.62	25.97	26.23	27.83	28.03

Table 2: Improvement in PSNR after denoising

Noise Level	DnCNN-B (paper)	DnCNN-B (ours)	DnCNNRes-B
$\sigma = 15$	+7.00	+7.18	+9.30
$\sigma = 25$	+8.98	+10.95	+11.09
$\sigma = 50$	+12.09	+13.68	+13.88

Another consideration when comparing these models is the execution time. Despite our DnCNNRes-B model having over 4x the number of trainable parameters, it only took on average about 58% longer for inference and noise subtraction to create a denoised image. On an Intel 2.1 GHz processor the DnCNN-B model took approximately 12 ms to to denoise a 321x481 image, while its big brother the DnCNNRes-B took approximately 19 ms. While this is a significant difference in processing time, the model could still process 30 frames at that resolution in about 570 ms. Given that most video is 30 FPS, the model should be able to denoise higher resolution video in real-time.

Results of our best algorithm, the DnCNNRes-B can be seen in the figures below. fig. 19 shows the clean image, the clean image with additive Gaussian noise, and the image after being denoised by the DnCNNRes-B algorithm on an image with distant features. fig. 20, shows the results of the same process on an image with near features.

(a) Clean Image

(b) Clean Image with Gaussian noise $\sigma = 25$

(c) Denoised image

Figure 19: Results of denoising on image with distant features.

(a) Clean Image

(b) Clean Image with Gaussian noise $\sigma = 25$

(c) Denoised image

Figure 20: Results of denoising on image with near features.

fig. 21 shows the results of the model when denoising an image corrupted with pepper noise on an image with distant features. fig. 22 show the results of the same process on an image with near features. If you observe the clean image in the 2nd set of examples you'll notice that there is an abundance of black spots in the sand, due to shadows, that resemble the pepper noise. The results of the denoising process show that the model did an exceptional job differentiating between the noise and the features in the clean image that resemble the noise.

(a) Clean Image

(b) Clean Image with 15% of pixels corrupted with pepper noise

(c) Denoised image

Figure 21: Results of denoising on image with distant features.

(a) Clean Image

(b) Clean Image with 15% of pixels corrupted with pepper noise

(c) Denoised image

Figure 22: Results of denoising on image with near features.

4.5 Conclusion

We were able to create and test 3 model types for image denoising with deep neural networks. The first model, the MLP, performed admirably on black and white images, given its non-spatial persevering structure. Unfortunately, the MLP was not able to learn to denoise color images. The DnCNN-B, a fully convolutional model, which allows for the spatial structure of an image to be maintained through out the network, did significantly better and had no problem denoising color images. Our implementation showed a nearly 15% PSNR improvement after denoising compared to the original DnCNN-B on the same benchmark dataset. And finally we produced a hybrid of the DnCNN-B and ResNet models which we dubbed DnCNNRes-B. By adding skip connections between every other convolutional layer in the model, we were able to successfully train a significantly deeper model than the original DnCNN-B. Our DnCNNRes-B model achieved a nearly 22% PSNR improvement after denoising compared to the original DnCNN-B on the same benchmark dataset. The results of our experiments show that not only are convolutional neural networks much better learning vision related functions, but given the right architecture, a significantly deeper model can achieve much better results on these types of tasks as originally demonstrated by the ResNet paper.

Acknowledgement

We would like to thank Professor Ted Willke for his instruction and helpful advice during this project. We would like to thank Professor Feng Liu for suggesting types of synthetic noise that resemble real world image noise. We would also like to thank Dr. Christof Teuscher and teuscher.:Lab for the use of the training equipment.

References

[1] H. C. Burger, C. J. Schuler, and S. Harmeling, "Image denoising: Can plain neural networks compete with bm3d?" in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, June 2012, pp. 2392–2399.

[2] 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, July 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

A Appendix

All code for this project along with the trained models and their results on the CBSD68 dataset can be found in this GitHub repository, Image Denoising with DNNs. The code is represented by an OpenSource MIT license, so please use as you see fit.

*** A big thanks to John Lipor for supplying the template that this one is based on! ***