# Exploring Convolutional Auto-encoders for Image Denoising.

Christos Psychalas

School of Informatics, Aristotle University of Thessaloniki

Contact: chrispsyc@yahoo.com

5 May, 2024

**Abstract**

In the domain of computer vision, denoising is the process of removing noise, which can take different forms from Gaussian to salt and pepper noise and more. This is a well known problem and many classic approaches have dealt with it, before the advent of Machine Learning, each with its advantages and limitations. In this work we explore the use of Auto-encoders, an unsupervised deep learning approach, that tries to reproduce an image to its output. This generative model can be used for image denoising as well, as the process of image reconstruction from the latent space is able to, implicitly, get rid of the noise. It is a strong technique as it does not need to map the noise, whose nature most of the times is random. The primary methodology involves developing and fine-tuning a CNN architecture using the CIFAR-10 dataset. The CNN model is trained to reduce noise levels in images sufficiently for a classifier to recognize key features, rather than aiming for perfect image reconstruction. As there is not a metric for generative models to compare to the initial image, a classic denoising metric is used as a proxy, the Peak Signal to Noise Ratio (PSNR). Finally, the results are evaluated using a Visual Transformer fine tuned on the CIFAR-10 dataset.

**Index Terms:** Auto-encoders, CNNs, Denoising, CIFAR-10.

## 1 Introduction

This study focuses on constructing a Convolutional Neural Network (CNN) architecture for image denoising [1], highlighting the process of investigating a model architecture and fine tuning it, which can be very cumbersome, as well as the power of Convolutional Neural Nets in tasks other than image classification, segmentation. Due to the limitation of resources, the whole process is applied on the CIFAR-10 dataset, which contains 10 classes and 50,000 images of low resolution, making the training on weaker machines easier. The approach followed in this work is not literature based, rather an experimentation, which can be easily seen following the steps presented later on. The back and forth steps highly designate this experimental nature, but also the difficulty in building something from scratch in the deep learning field without leveraging on the collective knowledge of the field. The main conclusion drawn from this process is to always try scaling your model step by step and not start building complex architectures, expecting to work better, although sometimes this is not enough as many parameters interfere. All the code and experiments can be found on the following GitHub repository, https://github.com/Xritsos/Denoising.

### 1.1 Paper Organisation

This report is structured as follows: in section 2 a brief introduction on Auto-encoders is given. In section 3, *Methods*, the noise creation and application process is presented along with the steps followed to build the final architecture. Later on, in section 4 the results of the denoising application are presented, closing with some conclusions on the whole process.

# 2 Background

The Auto-encoder is an unsupervised learning neural network that tries to generate its input, in this case an image, to its output. It consists of two parts an encoder, that reduces the input image to a latent space, compressing it to a smaller dimension [2]. Then, the decoder block tries to reconstruct the initial image from the compressed one in the latent space. The CNN Auto-encoders is a variation of the classic architecture, that focuses on using convolution layers on the encoder block and transposed convolution on the decoder block to reconstruct the input image. A useful application of Auto-encoders is the image compression performed by the encoder that can help reduce the size of the image, sent over, for example via internet, and then reconstructed using the decoder, making data transfer faster. But, Auto-encoders can implicitly perform denoising in images. Their image generation property make them suitable for generating clean images out of noisy ones, without focusing on mapping the noise in the initial image.

## 2.1 Literature Review

As already mentioned this study did not exploit any literature findings, thus this section remains very short. Nonetheless, the work on Auto-encoders is cited [3].

# 3 Methods

In this section details are provided on applying noise to CIFAR-10 images. Also, in section *3.2* the steps for choosing architecture are given in detail. Specifically, all the steps that are followed are presented, with the struggle of making model simpler and more complex each time. The complete tests can be found on the GitHub repository in file ***tests_fully.csv*** along with the file ***Notes_on_training.odt*** where each step performed was kept as a personal log.

## 3.1 Adding Noise

Image noise can take many forms, like gaussian, multiplicative, etc [4]. For the purpose of this study two different noises where added to each image or their combination. First of all, each time an image was loaded a random generator was applied to determine which kind of noise to apply, or which of their combination. All in all, there are four different noise types. More specifically, the first noise type (Type-I) is gaussian in nature, but instead of a zero mean different means and standard deviations were chosen and then added to the initial image. In case of the standalone case the mean value and the standard deviation chosen are 0.3 and 0.05, respectively. As seen below, in Figure 1 this combination added enough noise without totally destroying the image. The second noise type (Type-II) added resembles the gaussian, as it is additive in nature, but except for only taking values from a distribution and add them to the image, this noise first is multiplied with the values of the image and then added back to the image. For the standalone case a mean and a standard deviation of 0.8 and 0.5 were used. Except for these two types their two combinations were used as well. In the table below, the values for mean and standard deviation used in each case are presented.

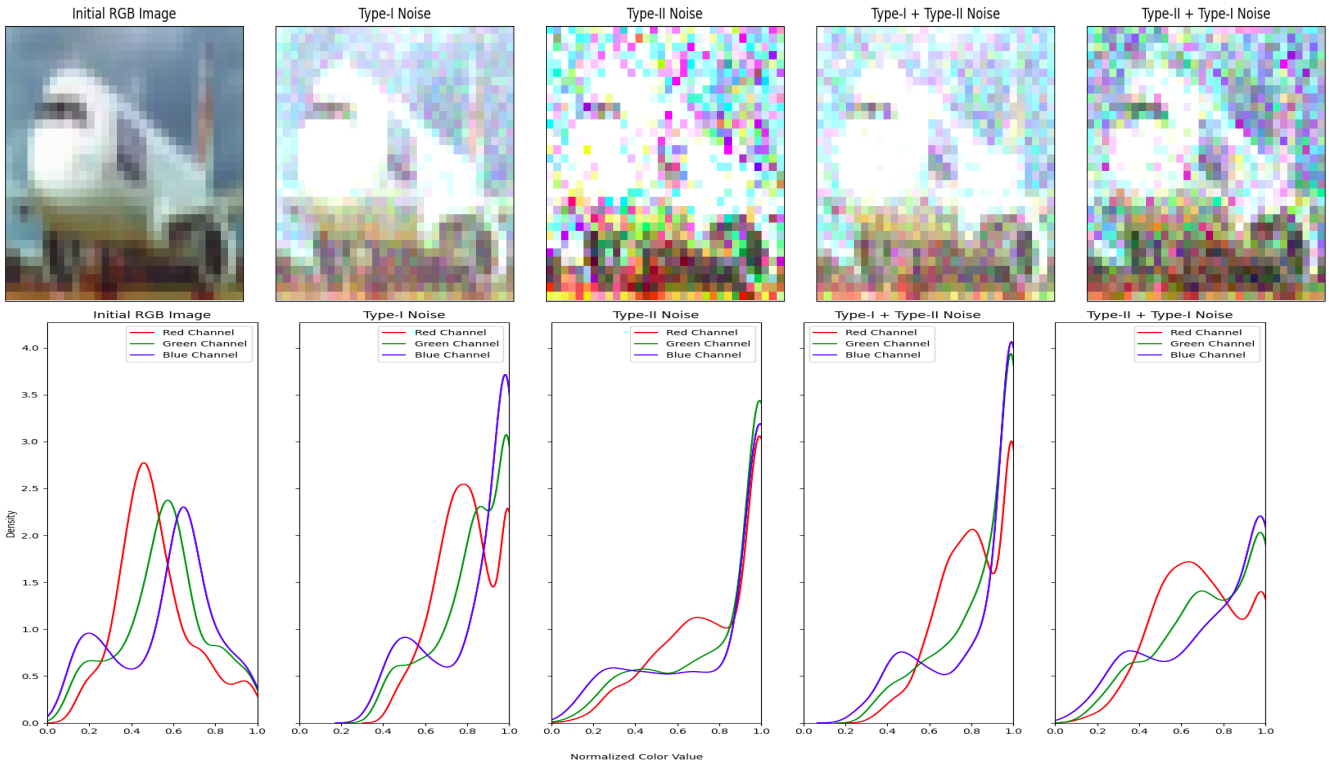


Figure 1: Image with different noise types and their distributions per channel.

Table 1: The values of mean and standard deviation for different noise application cases. [1]

| **Noise Type** | **Mean** | **Standard Deviation** |
|---|---|---|
| Type-I | 0.3 | 0.05 |
| Type-II (image values used) | 0.8 | 0.5 |
| Type-I + Type-II | 0.2 / 0.2 | 0.05 / 0.1 |
| Type-II + Type-I | 0.2 / 0.2 | 0.2 / 0.05 |

## 3.2 Dataset

The data used for the experiments are the $32 \times 32$ RGB images of the CIFAR-10 dataset. The data are acquired using the torchvision datasets, which provide the data in the $[0, 1]$ range. The module provides the test set and the size of the validation set can be set manually. All ten classes are balanced and almost equally represented in all sets. The train set includes $50,000$ images and the test set $10,000$ images. In the beginning, $45,000$ images were used as training set and the rest $5,000$ as the validation set. The volume of the images for training proved to be very big for the machine where the experiments were conducted, thus the intermediate split consists of $20,000$ training images and $5,000$ validation images. The test set was kept intact. For the final model training the split changed to $40,000$ training images and $10,000$ validation images. The final split used can be seen in the figure below.

[1] These values are the final values chosen after investigation.

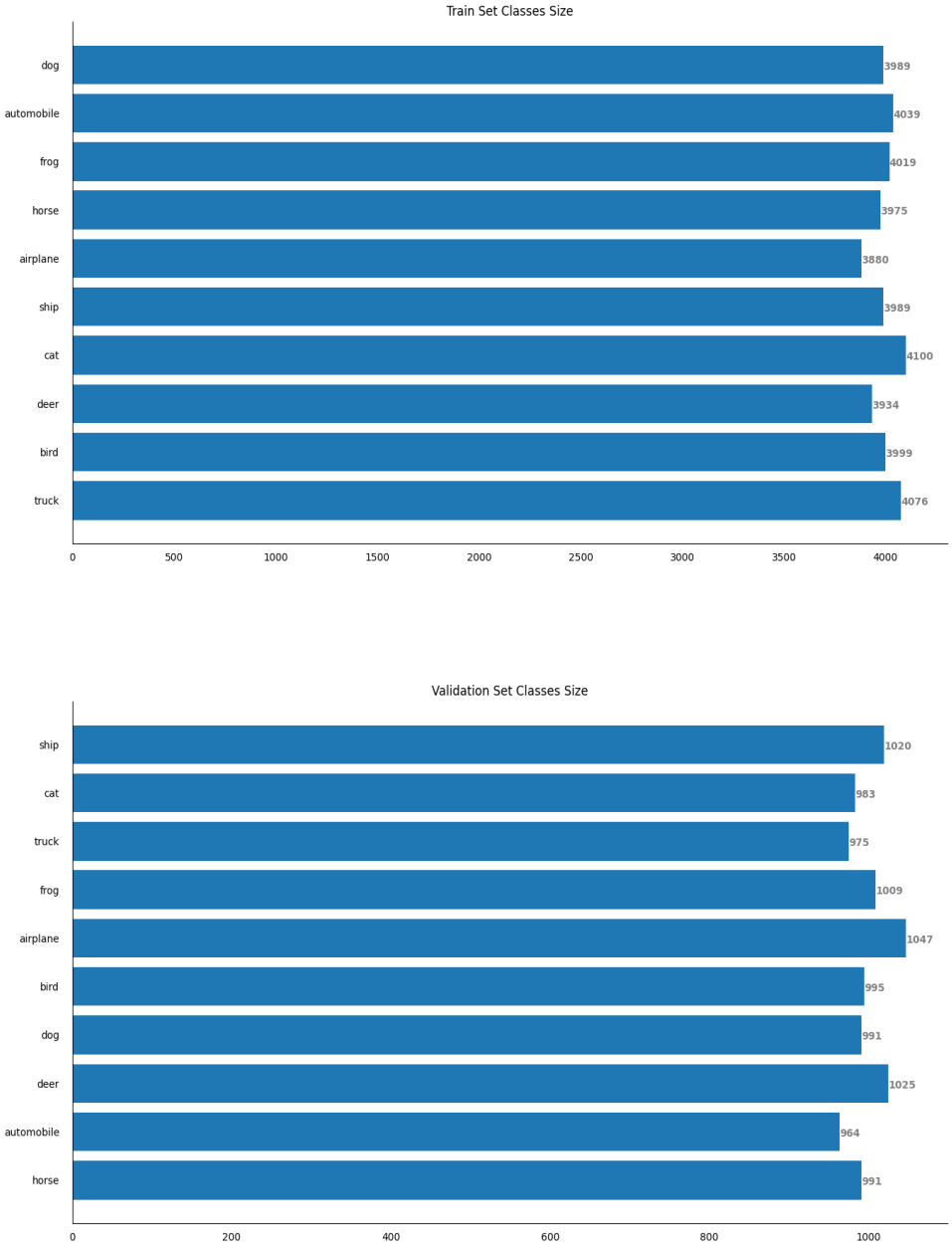


Figure 2: Balance of classes for train and validation sets.

## 3.3 Architecture Exploration

In this part the steps for concluding with an architecture are presented. Initially, the study started with a network comprising of fourteen layers (seven layers in the encoder and seven in the decoder) starting with high depth and leading to lower, as can be seen in the figure below.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 64, 32, 32] | 1,792 |
| Conv2d-2 | [-1, 64, 16, 16] | 16,448 |
| Conv2d-3 | [-1, 32, 16, 16] | 18,464 |
| Conv2d-4 | [-1, 32, 8, 8] | 4,128 |
| Conv2d-5 | [-1, 16, 8, 8] | 4,624 |
| Conv2d-6 | [-1, 16, 4, 4] | 1,040 |
| Conv2d-7 | [-1, 8, 4, 4] | 1,160 |
| ConvTranspose2d-8 | [-1, 16, 4, 4] | 1,168 |
| ConvTranspose2d-9 | [-1, 16, 8, 8] | 1,040 |
| ConvTranspose2d-10 | [-1, 32, 8, 8] | 4,640 |
| ConvTranspose2d-11 | [-1, 32, 16, 16] | 4,128 |
| ConvTranspose2d-12 | [-1, 64, 16, 16] | 18,496 |
| ConvTranspose2d-13 | [-1, 64, 32, 32] | 16,448 |
| ConvTranspose2d-14 | [-1, 3, 32, 32] | 1,731 |

Figure 3: First architecture tested (Model-A).

The intermediate layers get activated by ReLU and the output layer by sigmoid activation function. The loss function chosen is the Mean Square Error (MSE) and the optimizer is the widely used Adam, as provided by the pytorch package. In the first stage the above architecture is trained on the big split of $45,000$ images. In order to make sure that the decreasing loss actually makes the model learn, the Peak Signal to Noise Ratio (PSNR) is used, which is a common measure in image denoising. The problem here lies on the fact the images are generated rather than just having their noise removed. So, one should not expect high values of PSNR (in dB). After visual inspection it can be proven that increasing PSNR leads to better reconstructions. Another attempt was made to include a pre-trained classifier on the CIFAR-10 dataset, but the only one found was a Visual Transformer, whose inference time, with the current setup, was prohibitive. This model architecture manages to learn the dataset but the learning process is too slow, even when trying bigger learning rates. Moreover, the runtimes due to the large training set point to the direction of training data reduction, which is the case $(20,000)$ as mentioned in the previous section. Even with this reduction the model got stuck, not able to learn from a point on. The next step was to re-calculate the noise added to the images, leading to the final values given in Table 1.

Following the noise distribution alteration, the model hits a maximum of 0.0055 validation loss and its results (examined visually) are not satisfactory. To deal with it batch normalization layers are added, before activation functions [5], except for the last layer of the decoder. In the beginning, the batch normalization layers did not seem to fix the staleness of the validation loss. A series of alterations took place, like increasing feature depth on convolution layers, batch normalization layers removal and architectural revision by increasing feature depths instead of decreasing. Finally, the combination that yielded the most promising results was the one that makes use of batch normalization layers and feature depth decrease inversion. Trying to fine tune the model again the validation loss gets stuck and a series of tests were carried out, with the most prominent ones being increasing and decreasing the model's depth back and forth, until it was obvious that a shallower model was fitter.

A possible problem that was raised was that the validation loss was quite unstable. Another round of model architecture tweaking was performed, with back and forth layer removal. Increasing the batch size did not seem to fix the problem, but reducing the learning rate did. This came with the cost of the model getting stuck earlier in the training. There have been trials on more specific parameters of the optimizer such as weight_decay, that is model's weights regularization and the betas, which are responsible for the learning rate adaption. None of these fixed neither the validation loss fluctuations nor the loss being stuck, on the Model-B.

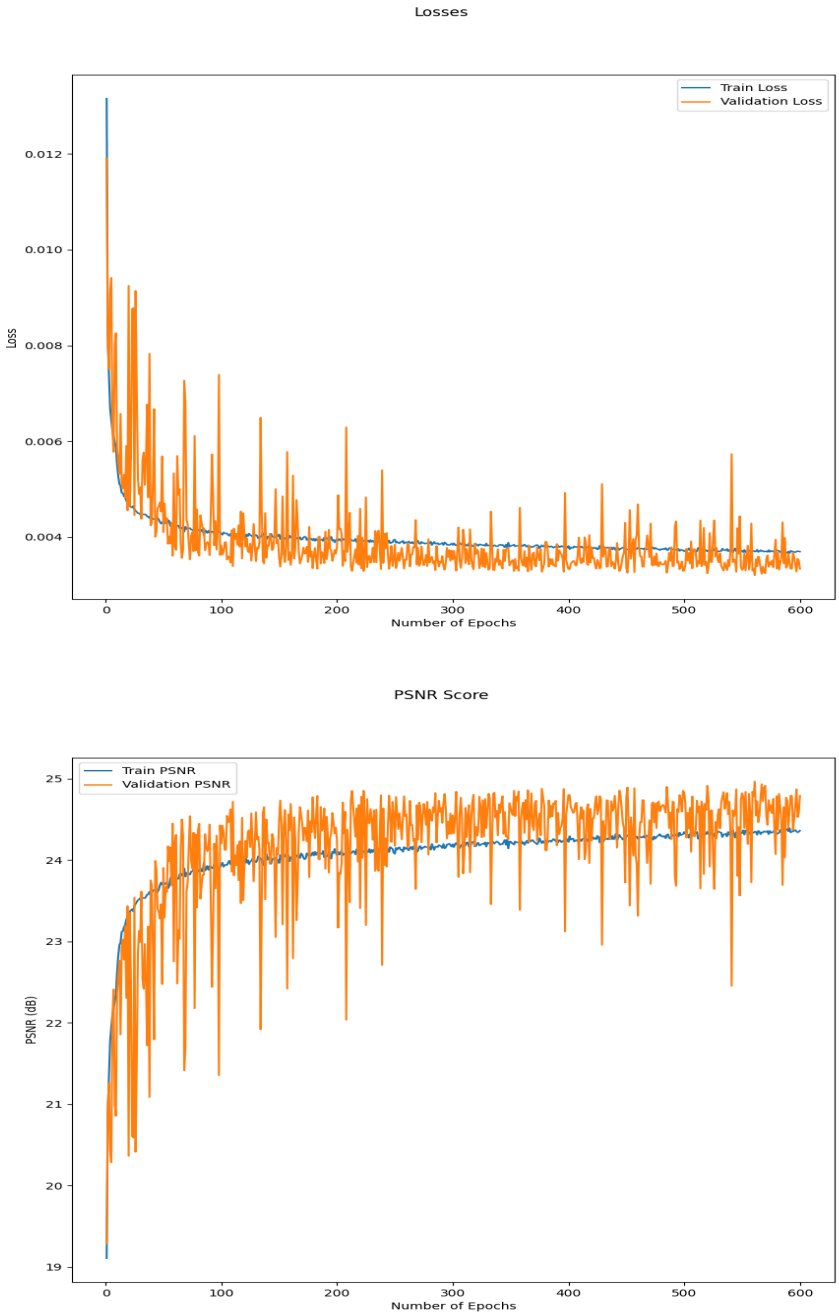


Figure 4: Validation loss and PSNR oscillations (test_id 35).

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 8, 32, 32] | 224 |
| BatchNorm2d-2 | [-1, 8, 32, 32] | 16 |
| Conv2d-3 | [-1, 8, 16, 16] | 264 |
| BatchNorm2d-4 | [-1, 8, 16, 16] | 16 |
| Conv2d-5 | [-1, 16, 16, 16] | 1,168 |
| BatchNorm2d-6 | [-1, 16, 16, 16] | 32 |
| Conv2d-7 | [-1, 16, 8, 8] | 1,040 |
| BatchNorm2d-8 | [-1, 16, 8, 8] | 32 |
| Conv2d-9 | [-1, 32, 8, 8] | 4,640 |
| BatchNorm2d-10 | [-1, 32, 8, 8] | 64 |
| Conv2d-11 | [-1, 32, 4, 4] | 4,128 |
| BatchNorm2d-12 | [-1, 32, 4, 4] | 64 |
| Conv2d-13 | [-1, 64, 4, 4] | 18,496 |
| BatchNorm2d-14 | [-1, 64, 4, 4] | 128 |
| Conv2d-15 | [-1, 64, 2, 2] | 16,448 |
| BatchNorm2d-16 | [-1, 64, 2, 2] | 128 |
| ConvTranspose2d-17 | [-1, 64, 4, 4] | 16,448 |
| BatchNorm2d-18 | [-1, 64, 4, 4] | 128 |
| ConvTranspose2d-19 | [-1, 32, 4, 4] | 18,464 |
| BatchNorm2d-20 | [-1, 32, 4, 4] | 64 |
| ConvTranspose2d-21 | [-1, 32, 8, 8] | 4,128 |
| BatchNorm2d-22 | [-1, 32, 8, 8] | 64 |
| ConvTranspose2d-23 | [-1, 16, 8, 8] | 4,624 |
| BatchNorm2d-24 | [-1, 16, 8, 8] | 32 |
| ConvTranspose2d-25 | [-1, 16, 16, 16] | 1,040 |
| BatchNorm2d-26 | [-1, 16, 16, 16] | 32 |
| ConvTranspose2d-27 | [-1, 8, 16, 16] | 1,160 |
| BatchNorm2d-28 | [-1, 8, 16, 16] | 16 |
| ConvTranspose2d-29 | [-1, 8, 32, 32] | 264 |
| BatchNorm2d-30 | [-1, 8, 32, 32] | 16 |
| ConvTranspose2d-31 | [-1, 3, 32, 32] | 219 |

Figure 5: Intermediate architecture (Model-B).

The Adam optimizer after many tests gave a validation loss of 0.00338, at test_id 27, that can be found on the repository in the ***tests_fully.csv*** file. The learning process of the model was very slow, so a different optimizer was tested, that is the classic Stochastic Gradient Descent, with a Nesterov momentum. Generally, this optimizer is said to be able to converge better, but at the cost of time. It must be noted that even with high learning rate the optimizer manages to steadily decrease the loss, while at the same time keeping the fluctuations of the validation loss very low. Although, SGD seems to converge smoothly, it took 600 epochs to nearly approach the validation loss that Adam managed with 100 epochs, even though with Adam many epochs passed with the validation loss going back and forth without actual changing. As a next step the best setup so far (Model-B) will is trained in many epochs, to see if the barrier of 0.00338 loss can be lifted. The model hits its limit at 0.00319 validation loss. Finally an alteration to this architecture is tested. Instead of adding or removing layers, we try to make the feature depth bigger. With this deeper feature dimension the model continues with decreasing its loss. On this final architecture we add all the data (test_id 38) and train it anew, reaching a final validation loss of 0.002.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| Conv2d-1 | [-1, 32, 32, 32] | 896 |
| BatchNorm2d-2 | [-1, 32, 32, 32] | 64 |
| Conv2d-3 | [-1, 32, 16, 16] | 4,128 |
| BatchNorm2d-4 | [-1, 32, 16, 16] | 64 |
| Conv2d-5 | [-1, 64, 16, 16] | 18,496 |
| BatchNorm2d-6 | [-1, 64, 16, 16] | 128 |
| Conv2d-7 | [-1, 64, 8, 8] | 16,448 |
| BatchNorm2d-8 | [-1, 64, 8, 8] | 128 |
| ConvTranspose2d-9 | [-1, 64, 16, 16] | 16,448 |
| BatchNorm2d-10 | [-1, 64, 16, 16] | 128 |
| ConvTranspose2d-11 | [-1, 32, 16, 16] | 18,464 |
| BatchNorm2d-12 | [-1, 32, 16, 16] | 64 |
| ConvTranspose2d-13 | [-1, 32, 32, 32] | 4,128 |
| BatchNorm2d-14 | [-1, 32, 32, 32] | 64 |
| ConvTranspose2d-15 | [-1, 3, 32, 32] | 867 |

Figure 6: Final architecture (Model-C).

# 4 Results

In this final section the results of the best model (Model-C) are presented and compared to the previous best model (Model-B). For this purpose a Visual Transformer is used that has been trained on the CIFAR-10 dataset as a proof of context that the model actually learned and did not get overfit. In the figure below we can see an example of the model's output on a noisy image. The output is very close to initial CIFAR-10 image as can be proven by the channel distributions before and after inference. It is true that in some cases the noise was too much for the model to handle, but given more feature depth the model should be able to learn this noise as well.

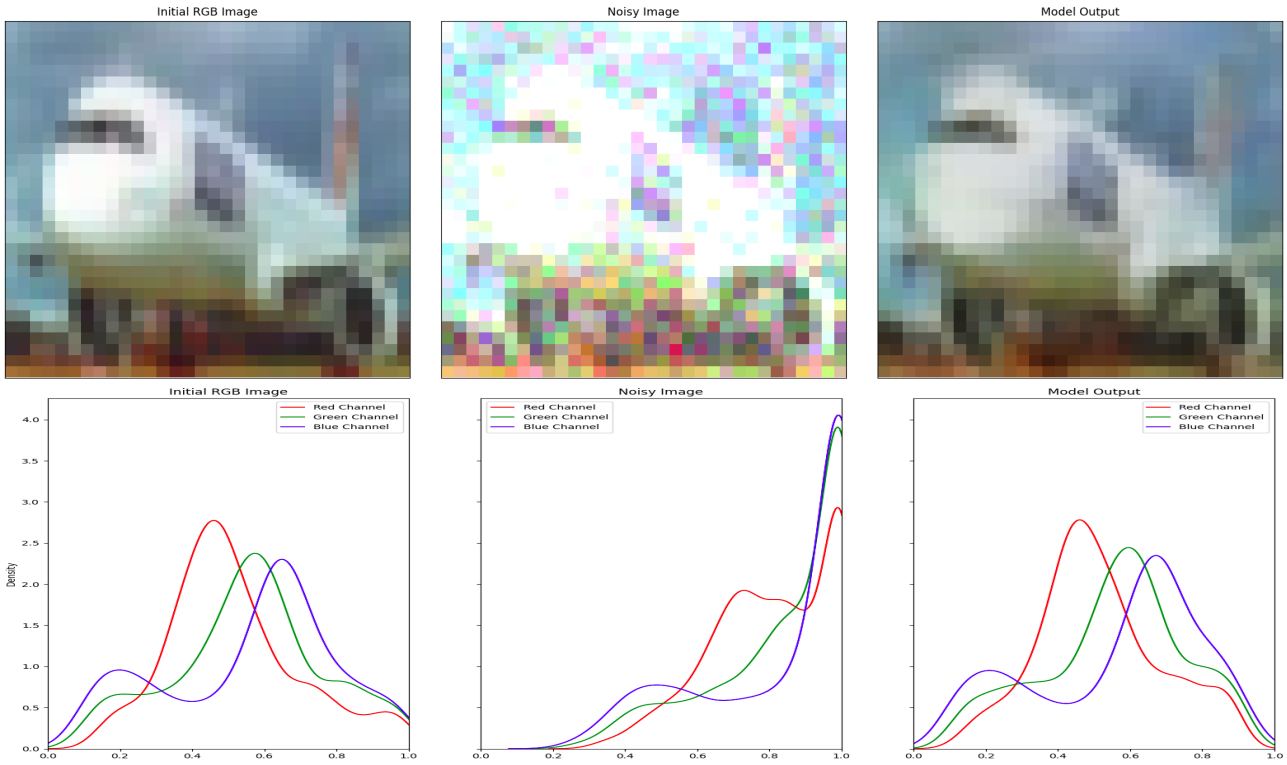


Figure 7: Final model's output to noisy image.

To ensure that the model did not overfit we tested it, along with the previous best model on the ViT classifier (https://huggingface.co/nateraw/vit-base-patch16-224-cifar10), which manages a $97,7\%$ Macro F1 score on the original CIFAR-10 dataset. In the table below we can see the final results.

Table 2: Results of different models using ViT trained on CIFAR-10.

| **Model** | **Val Loss** | **Val PSNR (dB)** | **Macro F1** | **Train/Val Size** |
|---|---|---|---|---|
| CIFAR-10 | - | - | $97,72\%$ | - |
| Model-B (test_id 35) | 0.003199 | 24.96 | $68.81\%$ | $20,000/5,000$ |
| Model-C (test_id 37) | 0.002193 | 26.6 | $80.99\%$ | $20,000/5,000$ |
| Model-C (test_id 38) | 0.002002 | 26.95 | $81.74\%$ | $40,000/10,000$ |

The increase in model's feature depth made it, significantly, generalize better as can be seen by the F1 score, although the increase in training size did not make any actual difference.

# 5 Conclusion

This process verified the rule that it is better to start with a simple architecture and slowly make changes. Nonetheless, it is also proven that the whole process, even if small steps are followed, is difficult to yield results. Even if the rule was followed and a slow increase in the model's depth was performed, as was the case, the target could not be reached. The crucial parameters that made a huge impact on making the model unstuck was, initially, the use of increasing feature depth and later on the use of batch normalization layers. But the most crucial parameter that helped overcome the loss stagnation, was the increase in feature depth. The current best model probably can learn even more and achieve a higher Macro F1 score if a further increase of its feature depth is performed. It is left for anyone who would like to try to make it even better.

# 6 References


[1] Anjuna CFX, YouTube Channel. Brief Introduction to Image Denoising. https://www.youtube.com/watch?v=qaccekBhS6E.

[2] Chirag Goyal, Analytics Vidhya. Image Denoising using AutoEncoders -A Beginner's Guide to Deep Learning Project. https://www.analyticsvidhya.com/blog/2021/07/image-denoising-using-autoencoders-a-beginners-guide-to-deep-learning-project/.

[3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning internal representations by error propagation*, page 318–362. MIT Press, Cambridge, MA, USA, 1986.

[4] Vijaysinh Lendave, Analytics India Magazine. Different Types of Noises and Image Denoising Methods. https://analyticsindiamag.com/a-guide-to-different-types-of-noises-and-image-denoising-methods/.

[5] Jason Brownlee, Machine Learning Mastery. A Gentle Introduction to Batch Normalization for Deep Neural Networks. https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/.


# A Appendix

## A.1 Runtimes

Table 3: Runtimes for different models.

| Model | Batch Size | Train/Val Size | Seconds per epoch |
|---|---|---|---|
| Model-A (test_id 1) | 32 | 45,000/5,000 | 57 |
| Model-A (test_id 5) | 64 | 45,000/5,000 | 47 |
| Model-B (test_id 35) | 32 | 20,000/5,000 | 15 |
| Model-B (test_id 35) | 64 | 20,000/5,000 | 10 |
| Model-B (test_id 35 + ViT) | 8 | 1,000/5,000 | 140 |
| Model-C (test_id 37) | 64 | 20,000/5,000 | 20 |
| Model-C (test_id 38) | 128 | 40,000/10,000 | 33 |

Specifications:
Processor: Intel® Core™ i7-8550U, 4 GHz (turbo mode)
GPU: Nvidia MX150, 2 GB
RAM: 16 GB
Operating System: Ubuntu 23.10