Comparative Analysis of PyTorch and TensorFlow for Image Noise Reduction Using Autoencoders

Max Zammit
Institute of Information & Communication Technology
Malta College or Arts, Science & Technology
Corradino Hill
Paola PLA 9032
{Max.Zammit.d102213@mcast.edu.mt}

***Abstract*—The reduction of image noise in image processing is important, and autoencoders have become popular tools for this purpose. This research aims to compare the effectiveness of PyTorch and TensorFlow, two widely used deep learning frameworks, in reducing image noise using autoencoder models. Autoencoders are neural networks that you can use to compress and reconstruct data. PyTorch and TensorFlow are two of the most commonly used deep learning frameworks that provide the necessary tools for implementing the auto-encoder models, so this research will evaluate PyTorch and TensorFlow's ability to see which is better to reduce image noise. We will also analyze in the discussion the advantages and disadvantages of using PyTorch versus TensorFlow for this task, including differences in ease of use, computational efficiency, and model flexibility. Overall, the goal of this analysis is to thoroughly compare two models PyTorch and TensorFlow to determine which method is preferable for users, so they can be able to choose the best framework for their particular requirements.**

***Index Terms*—MCAST, IICT, LaTeX, Project, Paper**

I. Introduction

Low-quality images, such as those with lots of noise or poor resolution, can make it difficult for image processing algorithms to function properly, producing inaccurate results and incorrect data interpretations. Some image processing techniques may include image segmentation, object detection, image analysis, and image recognition. [1] Noise is an undesirable signal, which generates a random variation of color or brightness values within an image. Noise can cause the color or brightness values of an image to fluctuate at random, making it an unwanted signal. Noise can produce undesirable results such as unrealistic edges, ignored lines/corners, and blurry objects, making manipulating an image more difficult. Noise may be collected during the process of acquiring or transmitting an image. The main objective is to compare two different autoencoder models with the same data set to see which improves images' visual quality and clarity by removing unwanted and distracting noise. The motivation behind this research is to explore new methods and compare the results of existing techniques to see the differences between more efficient and effective denoising techniques. The autoencoder concept is used in this research to clean up the degraded image's noise. [1] An artificial neural network with the goal of learning an encoding for a set of data is known as an autoencoder. The autoencoder attempts to produce a clean image that is as close to its original input as possible after learning noise from training the images.

II. Literature Review

Autoencoders were first used for image denoising in 2006 [2]. In 2010 these autoencoders for image denoising started gaining popularity and were being used to remove noise from medical images (X-rays, Magnetic Resonance Imaging MRI, ultrasounds, etc) [2]. Autoencoders have shown good results on a number of benchmark data sets for image denoising, with peak signal-to-noise ratio (PSNR) values exceeding 35 dB for noise levels ranging from 10% to 50%. Auto-encoder-based techniques have also been shown to be efficient without the need for explicit modeling. [3]. This enables the model to learn the underlying structure of the image data and remove the noise directly from the input data. Thanks to deep learning approaches, some work successfully combines feature learning and clustering into a unified framework that can directly cluster original images with even higher performance. "Deep Clustering with Convolutional Autoencoders": *Thanks to deep learning approaches, some work successfully combines feature learning and clustering into a unified framework that can directly cluster original images with even higher performance. [4]*

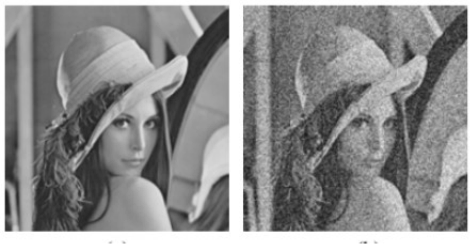
A. Data Set

The EuroSAT dataset [4] consists of images taken from satellites, for 10 different land use and land cover classes. The entire dataset consists of 27,000 labeled satellite images that are RGB and have a fixed size of 64x64 pixels. This dataset has a balanced distribution, diverse classes, and challenging variations which makes it suitable for assessing the performance of the autoencoders.

Figure 1: EuroSAT Data set

B. Adding Noise to Images

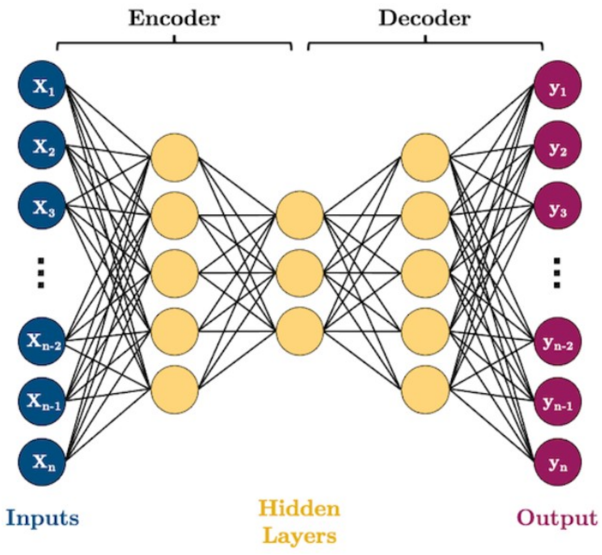
Figure 2: Data Preprocessing [5].



[5] Gaussian Noise is a statistical noise that has a probability density function of the zero mean normal distribution. To denoise the images, the images in the data set had to contain noise, so for the purpose of this project, Gaussian Noise will be used. Since most data observed in the real world have a Gaussian distribution of noise, the decision to choose this type of noise has been made for the purposes of this experiment.

C. Autoencoder Convolutional Layers

Figure 3: Autoencoder Encoder and Decoder Layers



In this paper [6], the authors propose an adaptive multi-level residual autoencoder for image denoising, which consists of three parts: encoding, decoding, and denoising.

[7] The encoding step has generated a compressed representation of the input image by extracting the features from the noisy image. In this study, the authors extract hierarchical features at various levels using a multi-level residual network that includes multiple residual blocks.

[7] The image was recreated using the decoding step from the compressed representation. For decoding, the authors employ a deconvolutional neural network that has been trained to produce an image that is as similar to the original as possible.

[7] The denoising step is created in the encoding step. The denoising step uses the input image's noise and removes it. The authors suggest an adaptive denoising module that, depending on the input image's noise level, can be adaptive to change how much noise is removed at each level of the multi-level residual network.

D. Evaluation of study comparing different models with the same dataset

Table 1 [8] Comparisons of different methods with θ = 25 for image denoising.

Methods	**PSNR**	**Dataset**
BM3D	28.57	BSD68
WNNM	28.83	BSD68
TNRD	28.92	BSD68
DnCNN	29.23	BSD68
FFDNet	29.19	BSD68
IRCNN	29.15	BSD68
DDRN	29.18	BSD68

1) **Methods** - This refers to the different deep learning methods which outperform the conventional methods.
2) **Peak Signal-to-Noise Ratio (PSNR)** - This refers to the expression for the ratio between the maximum possible value (power) of a signal and the power of distorting noise that affects the quality of its representation.
3) **Dataset** - This refers to the dataset which was used in the study which is BSD68.
4) **θ = 25** - This refers to a set of parameter value.

The deep learning technologies for image denoising that have been proposed recently were reviewed and summarized in a survey study on deep learning for image denoising [5]. According to Table 1, the deep learning methods such as BM3D, WNNM, TNRD, DnCNN, FFDNet, IRCNN, and DDRN outperform the conventional methods, with DnCNN achieving the highest PSNR score of 29.23. This suggests that deep learning-based methods are more effective for image denoising, and DnCNN is particularly successful. However, the performance of these methods may vary depending on the dataset and level of noise in the images.

III. Research Methodology

By using these autoencoders, we believe that it could be beneficial for image classification because it could affect the accuracy of any analysis or data processing steps. It is essential to ensure that images are as clear as possible before processing or analyzing to achieve the best results possible. The preprocessing steps such as noise reduction improve the image quality and facilitate further analysis for image classification.

With this research, the aim is to compare the results of two autoencoders with the same dataset, same model, same training, and validate to see the potential difference in results and analyze any possible improvements that could be done. Our objectives are to see:

1) What was the difference in results between the two models and what the difference is?
2) Which model was easier to build and use?
3) Can the PyTorch model will be more effective in handling images with complex structures, due to its flexible neural network architecture?

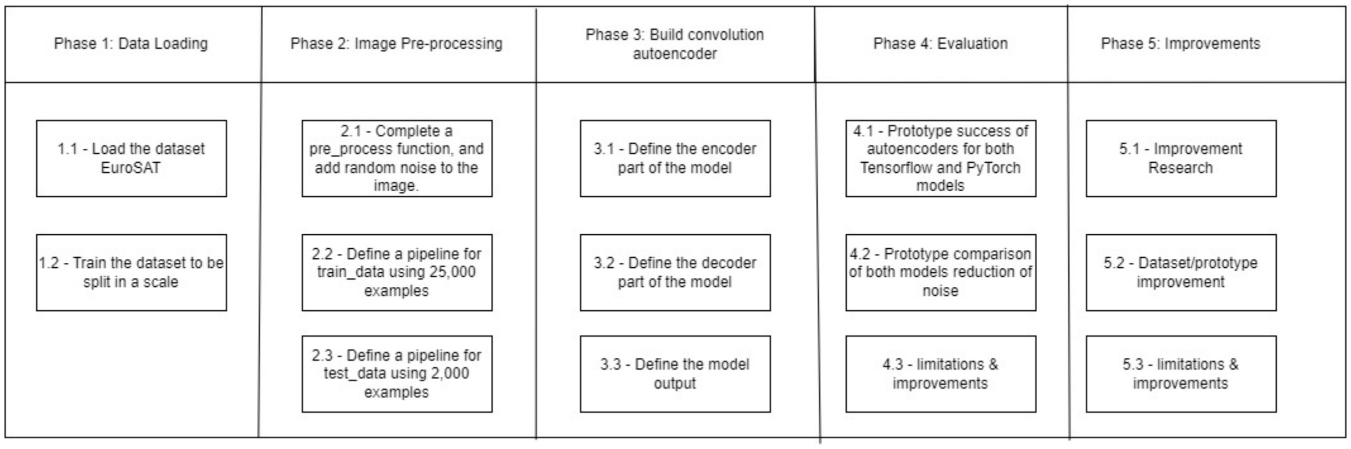

Phase 1: Data Loading
Phase 2: Image Pre-processing
Phase 3: Build convolution autoencoder
Phase 4: Evaluation
Phase 5: Improvements
1.1 - Load the dataset EuroSAT
1.2 - Train the dataset to be split in a scale
2.1 - Complete a pre_process function, and add random noise to the image.
2.2 - Define a pipeline for train_data using 25,000 examples
2.3 - Define a pipeline for test_data using 2,000 examples
3.1 - Define the encoder part of the model
3.2 - Define the decoder part of the model
3.3 - Define the model output
4.1 - Prototype success of autoencoders for both Tensorflow and PyTorch models
4.2 - Prototype comparison of both models reduction of noise
4.3 - limitations & improvements
5.1 - Improvement Research
5.2 - Dataset/prototype improvement
5.3 - limitations & improvements

Fig. 1. Research Pipeline

A. Data Loading - There difference when loading the dataset in Tensorflow vs Pytorch

The Tensorflow datasets module automatically downloads, unzips, and prepares metadata information. The Pytorch dataset module does not work for this particular dataset, so 'wget' in the os.systems module of Python was used to download the data into the root directory. Then the command line prompts you to unzip and prepare the data. While PyTorch requires user-written commands to convert the data type into a tensor, the Tensorflow module can automatically handle numpy arrays and convert them to tensors. The Dataloader module of Pytorch has to be used in Pytorch to split the data into batches and it also helps to save RAM memory.

B. Data Prepossessing

For the purposes of this study, the dataset needs to initially contain noise in order to denoise a picture.

For the purpose of this project, Gaussian Noise is added to the data set with zero mean and 0.2 standard deviations. For this research, Gaussian noise was used because it is typically the most common for the data observed in the actual world.

The dataset images in the Tensorflow and PyTorch models after adding Gaussian Noise look sufficiently pixelated but still have room for denoising since only 0.2 standard deviations were added.

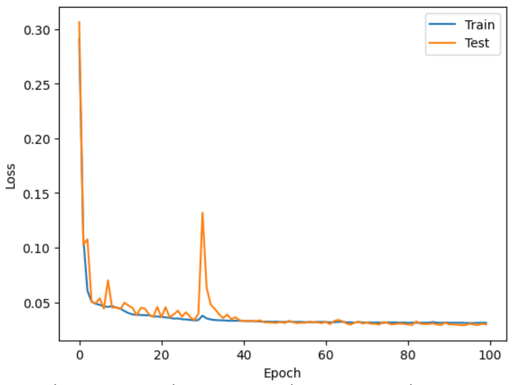
C. Autoencoder Model Building

With this research, the aim is to build two autoencoder models using both Tensorflow and PyTorch to analyze which model could be more advantageous to use for image denoising. The architecture of the model used for this project contains 3 convolution layers followed by 3 transpose convolution layers. A model with the same architecture is built in both TensorFlow and PyTorch. A few differences that one can observe is that with Keras. In the sequential module of TensorFlow, the encoder and decoder can be written in the same object. In PyTorch, the Autoencoder object had to be written using Python 'classes' and it isn't required to compile the model. Adam optimizer and MAE loss functions are used to train the model. The model with 3 layers of convolution and transposed convolution suits this data as the image size is relatively small (64 X 64 X 3).

D. Training and Validation

Both models were trained using Conv2D layers in the encoder and Conv2DTranspose layers in the decoder. Both models were trained for 100 epochs and the results of the models were plotted with the same layout. These are the results for both models

Figure 4: Results of the training and validation losses for the Tensorflow Model



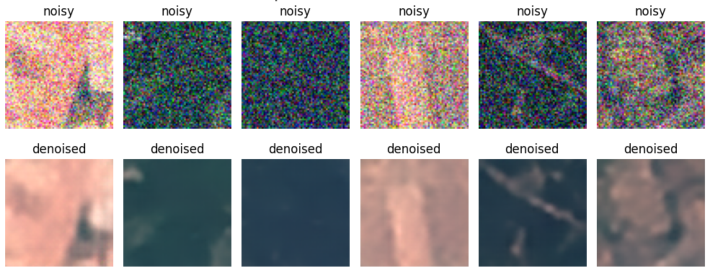
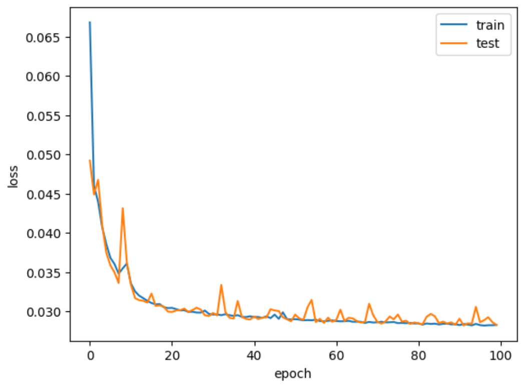
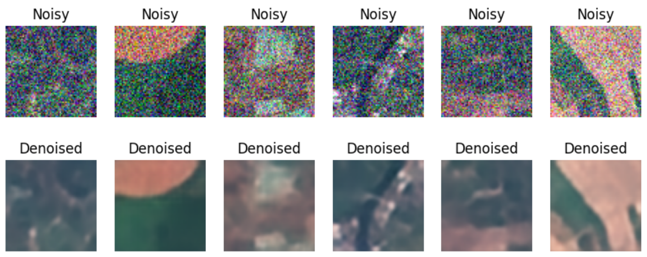


Figure 5: Results of the training and validation losses for the PyTorch Model





IV. Findings & Discussion of Results

Out of 27000 images, 25000 were used for training, and 2000 are used for testing and validation. Results show that the denoising architecture for both models worked well as shown in Figures 4 and 5.

Tensorflow had a module that automatically handled numpy arrays and converted them to tensors, whereas when importing the dataset into PyTorch, we had to manually write commands to convert the data type into a tensor. As a result, loading the dataset in Tensorflow was simpler than loading it in PyTorch. The PyTorch data loader module split the dataset into batches, which in the end, helped save RAM memory even though it made it simpler to import the dataset into Tensorflow.

There are some clear peaks in the validation curve of the Tensorflow model, whereas the peaks are less significant in the PyTorch model. So the loss was more gradually decreased in PyTorch model indicating better training. The outputs from both the models on test data are very similar and the PyTorch model also had slightly better training.

The initial loss (epoch 0) value for the PyTorch model is significantly lower than the loss value in the Tensorflow model, which indicates that the training load might have been initialized better in PyTorch and hence a little better performance overall.

There was a significant training loss till around 20 epoch, and after that, the loss remained almost the same till the end. This means that at least 30 epoch is required to run the model and achieve the best results.

V. Conclusion

In conclusion, both PyTorch and TensorFlow demonstrated effectiveness in reducing image noise using autoencoders while PyTorch showed a slight increase in performance and could save more RAM while training the model. It is noted that around 30 epochs were required to achieve the best results, as there was a significant decrease in training loss until around epoch 20, followed by a plateau. While the TensorFlow model exhibited clearer peaks in the validation curve, the PyTorch model showed better training performance overall. The choice between the two frameworks may depend on factors such as ease of use, computational efficiency, and model flexibility, which were not explicitly addressed in this paper. Further analysis and comparison of these factors would be valuable for determining the preferable framework based on specific requirements.

Although the autoencoder models in this study provided valid results, we recommend further study in the field of using autoencoders for image noise reduction.

An improvement in the models could possibly be adding different type of Gaussian noise, for example using Salt-and-Pepper noise or Poisson noise, and adapting the models to handle these specific types of noises to see which will lead to more effective denoising results.

Another improvement could be implementing different loss functions, such as mean squared error or perceptual loss, to see which could enhance the quality of denoised images produced by the autoencoder models.

Also, experimenting with increasing the depth or adding additional layers to the autoencoder models could potentially result in improved image-denoising performance.

References

[1] D. . A. M. Bajaj, Komal Singh, “Autoencoders based deep learner for image denoising,” *Procedia Computer Science.*, pp. 171. 1535–1541, 2020.

[2] L. Gondara, “Medical image denoising using convolutional denoising autoencoders,” *article submission*, p. 1608.04667v2, 2016.

[3] H. C. Burger, C. J. Schuler, and S. Harmeling, “Image denoising: Can plain neural networks compete with bm3d?” pp. 2392–2399, 2012.

[4] P. Helber, B. Bischke, A. Dengel, and D. Borth, “Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification,” *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 12, no. 7, pp. 2217–2226, 2019.

[5] L. F. Chunwei Tian, Yong Xu and K. Yan, “Deep learning for image denoising: A survey,” 2018.

[6] C. Song, S. Sudirman, and M. Merabti, “A spatial and frequency domain analysis of the effect of removal attacks on digital image watermarksǁ,” 01 2010.

[7] N. Shah and A. Ganatra, “Comparative study of autoencoders-its types and application,” pp. 175–180, 2022.

[8] X. Guo, X. Liu, E. Zhu, and J. Yin, “Deep clustering with convolutional autoencoders,” pp. 373–382, 10 2017.