19CSCI33H

Artificial Neural Networks

Final Report Phase 2

Student Name	Student ID
Karim Abdel El Wahab	159773
Ayah Abo El-Rejal	153509

1. Introduction

Contribution: All work was done by both students simultaneously.

a. Actuality of the chosen topic

Autoencoders are a unique type of neural networks. They are special in a way that they learn the way the data is coded in an unsupervised way. The aim of an autoencoder is to learn how the data is represented. This work will implement a special type of autoencoder called Denoising Autoencoder. The Denoising Autoencoder learns the representation of data and learns how to represent the data without the noise associated with it. For example, if an image has a lot of artifacts from other sources of noise. The Denoising Autoencoder works to ignore these artifacts.

b. Motivation to choose it

Denoising images is a vital pre-processing step in many fields. The images can receive a lot of noise from many factors. Factors like light and camera quality can affect how much noise exists. Therefore, images require more processing to remove these artifacts. This neural network can be beneficial for the medical field. Many deep learning algorithms were proposed to solve this problem, but none proved efficient. Some of them required a lot of data to train on and others required high computational costs. It was proved that denoising autoencoders produced good performance results using only 300 images for training in [1].

c. Main ideas, hypothesis, which are implemented and tested

In this work, the Denoising Autoencoder will be implemented to learn the representation of a dataset data and learn how to denoise images. Different hyperparameters like optimizers, number of hidden layers, etc. will be tested in this work. The most optimal model will be highlighted and diagnosed alongside its learning curves. The code uses the BSDS500 Dataset which consists of 300 images training/validation and 200 images for testing. We will be using Dataset Image Augmentation to increase this number to 1800 images for training, 400 images for validation and 400 for testing. Which roughly equals a percentage of 70% of images to training, 15% to validation and 15% to testing. The code is run on Google Colab and Keras is used alongside matplotlib and Tensorboard for visualization.

2. Data quality and pre-processing

Data Choice:

We chose the BSDS500 Dataset for our Denoising Autoencoder. We found it suitable as there is a different number of objects and environments present, as well as the number of dataset images is not too big or too small to overfit. We chose to do dataset augmentation to increase our dataset number from 500 to 2600 total.

Link to the original dataset is found here in [2].

Data Cleaning and normalization:

We checked the dataset for missing or unwanted images, but we did not find any images to clean. We added noise to the training set so that we have both a noisy training set and a non-noisy image training set. We have chosen to normalize our dataset to a set standard. The standard was that all images would be RGB and that they are 400x400 in size.

3. Theoretical background for the chosen architecture including the underlying statistics

Autoencoders were first introduced in the 1980s by Hinton and the PDP group (Rumelhart et al., 1986) to address the problem of "backpropagation without a teacher", by using the input data as the teacher.Together with Hebbian learning rules (Hebb, 1949; Oja, 1982), autoencoders provide one of the fundamental paradigms for unsupervised learning and for beginning to address the mystery of how synaptic changes induced by local biochemical events can be coordinated in a self-organized manner to produce global learning and intelligent behavior

Autoencoders are simple learning circuits which aim to transform inputs into outputs with the least possible amount of distortion and it is also a type of unsupervised feed forward neural deep learning techniques. They do not need explicit labels to train on. The output is like the input in structure but different in representation. They were introduced by Hinton in the 1980s to provide a solution for the "backpropagation without a teacher" problem. Autoencoder compress the input into latent space representation which is a lower dimensional code and then reconstruct the output from this representation form [3]. The output is the reconstructed input and it is usually blurry and of lower quality. This happens as a result of lost information during the compression stage. Denoising Autoencoders can be used for feature extraction, removing noise from pictures, generating images with higher resolution and dimensionality reduction. The Denoising Autoencoder is used to randomly corrupt the input by introducing noise and then it must reconstruct it. They are simply MLPs that are trained to denoise. It is an advanced version of the standard Autoencoders that tries to reduce the risk of learning the identity function. Specific properties of the proposed model are, Data specific: they only compress data that is like what they were previously trained on [4]. This is because they learn features that are specific only to the training set used. Moreover, they are Lossy; the output will be a degraded representation and not be the same as the input fed [4].

The initial image ($\boldsymbol{x}$) is corrupted into being a noisy image representation ($\tilde{\boldsymbol{x}}$).

Then, the corrupted image ($\tilde{\boldsymbol{x}}$) is mapped to a hidden representation which is made using the same autoencoder process**.**

$$\boldsymbol{h} = f_\theta(\tilde{\boldsymbol{x}}) = s(\boldsymbol{W}\tilde{\boldsymbol{x}} + \boldsymbol{b}).$$

Using the hidden representation, the output image is reconstructed.

$$z = g_{\theta'}(\mathbf{h}).$$

Then the error is calculated to calculate the difference between the reconstruction and original input image. So, the error between Z and X is calculated. The purpose of the statistics and underlying process is to try to minimize the error.

4. Implemented variations - justification

Several tests were made using different settings of hyperparameters. Most of the runs were made using the structure illustrated in the ***model.png*** figure.
The training/testing required 25GB of RAM in Google Collab using the free RAM upgrade Collab gives.

Different set of optimizers were tested like: **ADAM**, **RMSPROP**, **SGD**, **SGD + Nesterov**
Different set of loss functions were tested like **mean_squared_error**, **mean_squared_logarithmic_error**, **mean_absolute_error**

Each pair of optimizer and loss function were tested. Only a sample of runs is shown in the results below.

We used ADAM and RMSPROP for their known ability to work well in convolutional networks. The SGD and its improvement SGD + Nesterov were tested for comparison with the ADAM and RMSPROP outputs. As for the loss functions, it was a matter of fine tuning and testing each loss function with each optimizer to find the most suitable. We did not use classification optimizers like binary_crossentropy as they are only used in classifiers.

The layers consist of one input layer, 5 convolutional layers and several activation functions, batch normalization and up sampling layers. The encoder part of the structure decompresses the image, so it starts with a bigger number of hidden layers then as the encoder goes on the number of hidden layers are decreased. After the representation is made. The decoder then up-samples the representation back in the next layers. For more information about the exact structure of the layers, refer to ***model.png***.

Convolutional Layers Representation: **{conv2d_1, conv2d_2, conv2d_3, conv2d_4, conv2d_5}**

The testing first started with **{64, 32, 32, 64, 3}** nodes at convolutional layers but, after further representational bottleneck was found. The compression was too much for **400x400** images such that the images started to lose representational data. Therefore, the number of nodes were increased to **{100, 50, 50, 100, 3}**. This increased the RAM usage required, but there was less data loss from compression and the accuracies increased.

5. Results of training and validation

All Runs were made on:
Batch size: 16

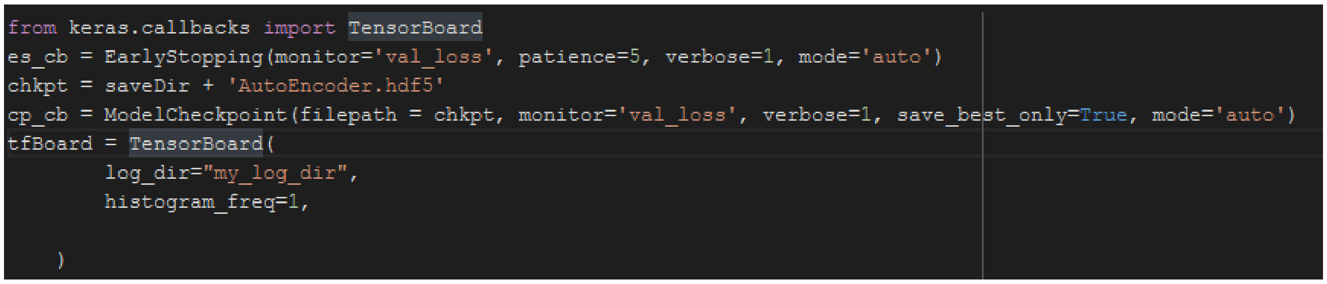
Epochs: 100

1800 training, 400 validation and 400 testing samples

 0.2 noise factor

Google Collab 25GB free update.

Callback functions used:

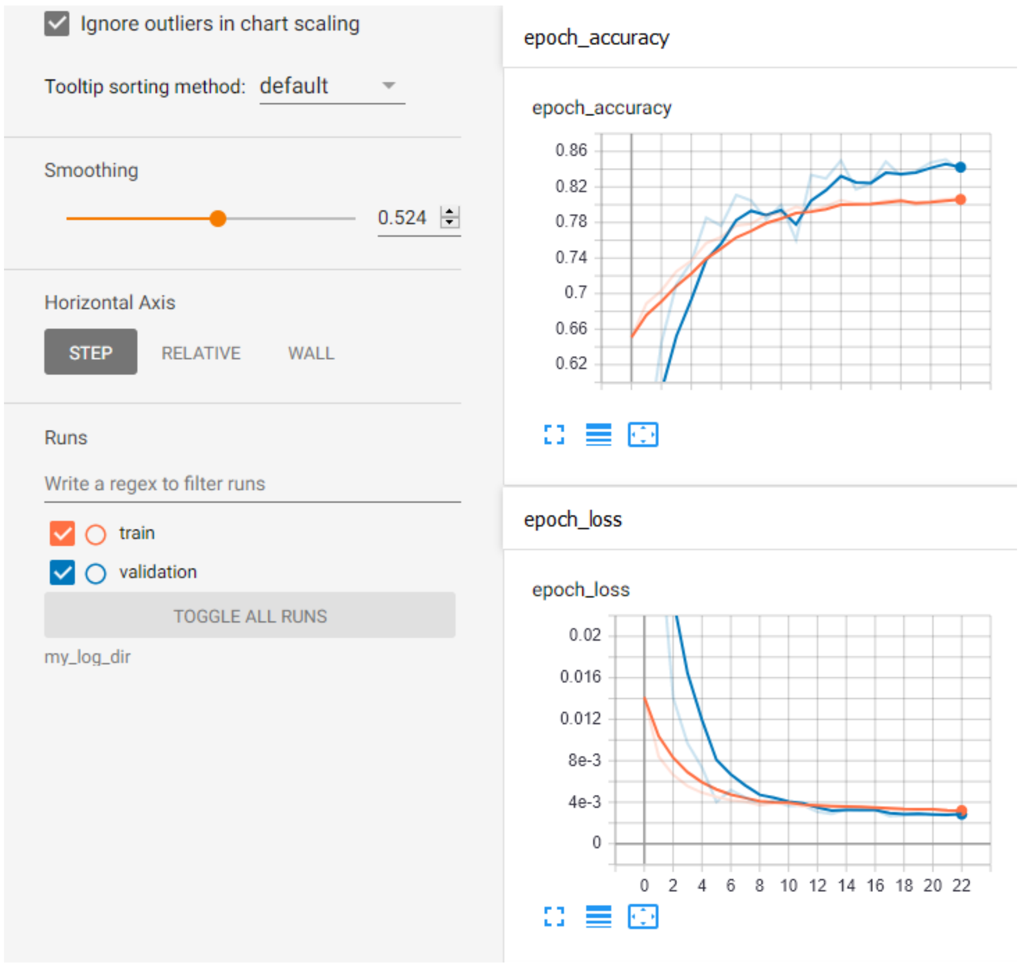


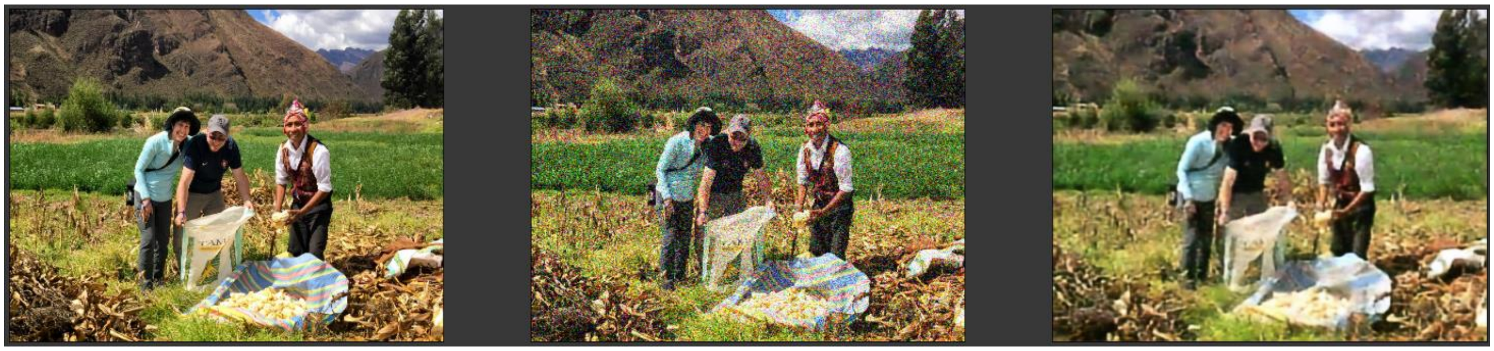
```
from keras.callbacks import TensorBoard
es_cb = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='auto')
chkpt = saveDir + 'AutoEncoder.hdf5'
cp_cb = ModelCheckpoint(filepath = chkpt, monitor='val_loss', verbose=1, save_best_only=True, mode='auto')
tfBoard = TensorBoard(
        log_dir="my_log_dir",
        histogram_freq=1,

    )
```

Run Configuration	Learning Rate	Early Stop (epochs)	Loss	Accuracy	MAE	Val_loss	Val_acc	Val_MAE
ADAM/ MSE	0.001	23/100	0.0033	0.8044	0.0431	0.0027	0.8485	0.0364
RMSPROP/ MSE	0.001	21/100	0.0031	0.8101	0.0421	0.0025	0.8207	0.0366
NAG/ MSE	0.01	100/100	0.0062	0.7315	0.0589	0.0052	0.7810	0.0517
ADAM/ MSLE	0.001	10/100	0.0019	0.7802	0.0464	0.0020	0.7957	0.0474

<u>Run 1:</u>

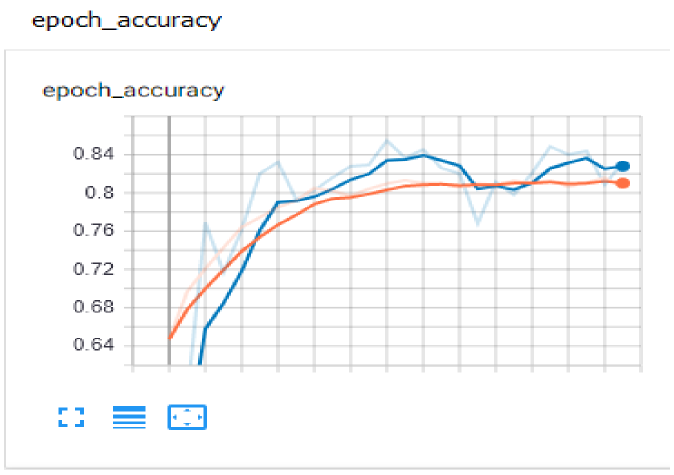


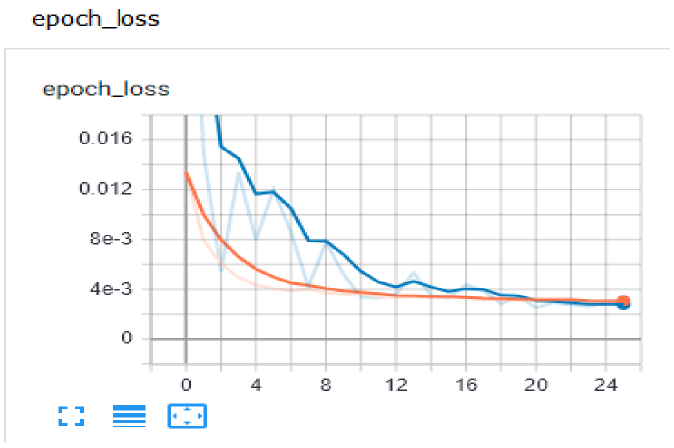


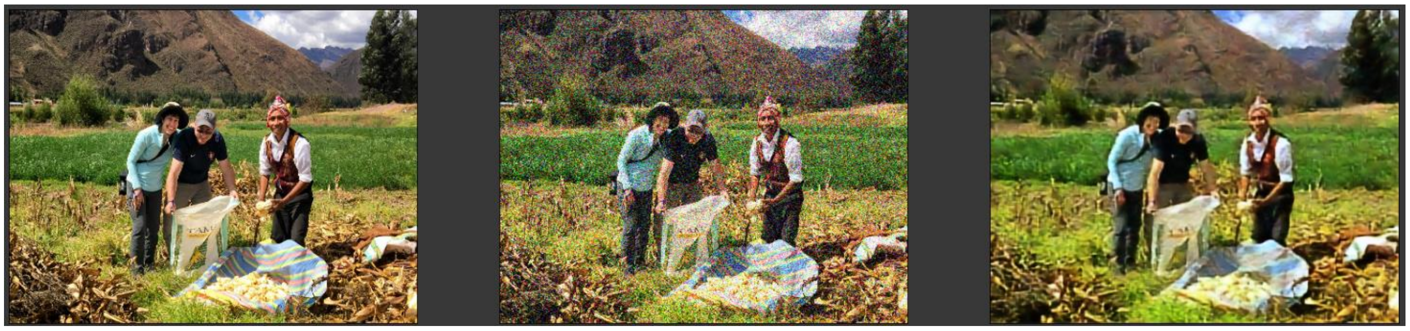
Original Noisy Deconstructed

The results are good. There is some loss from compression and noise, but the result is overall good. The graph is showing little overfitting which is not bad.

Run 2:



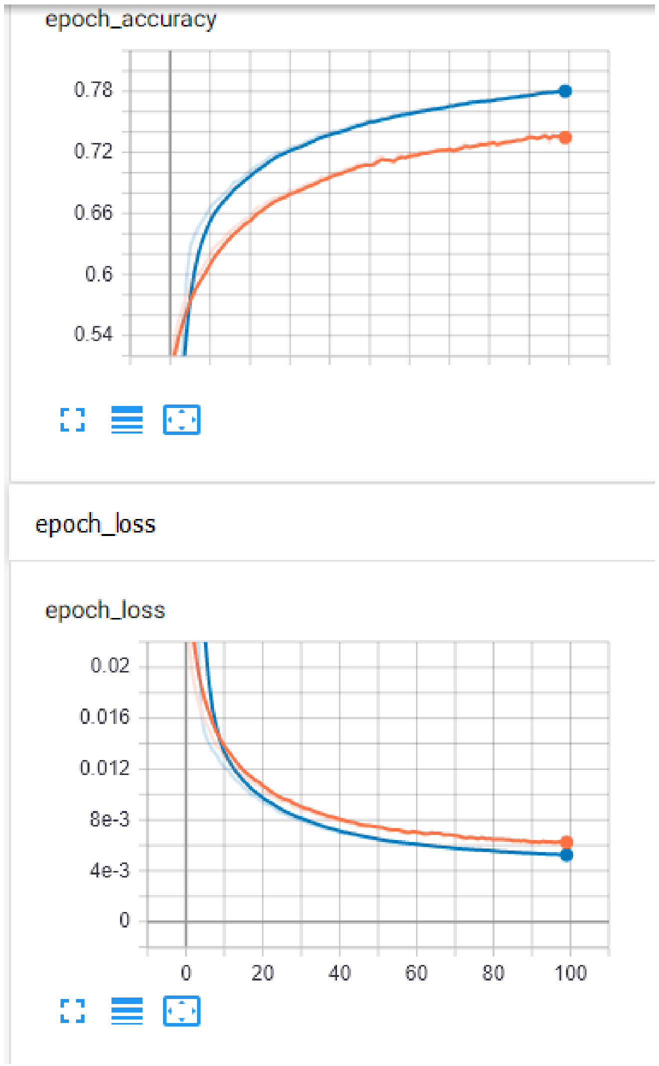


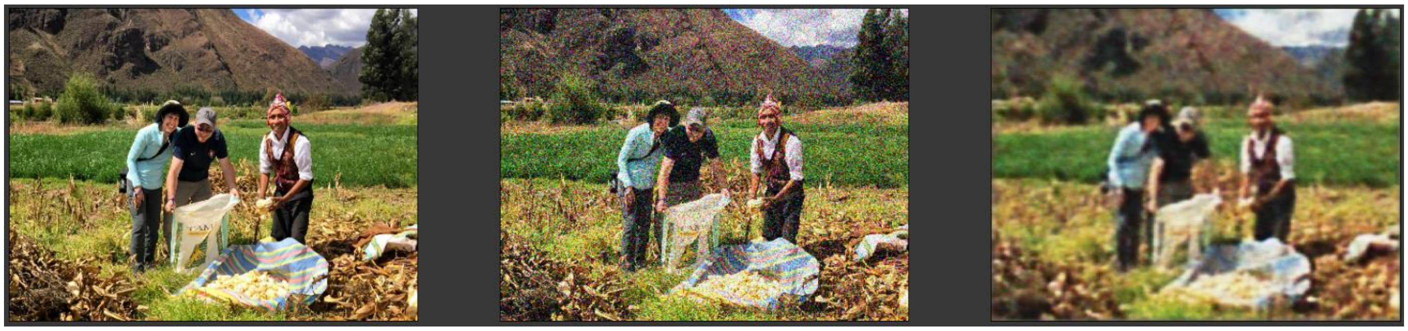


Original Noisy Deconstructed

The results are also good. Very similar to ADAM output. But the Accuracy of training and validation is closer.

Run 3:

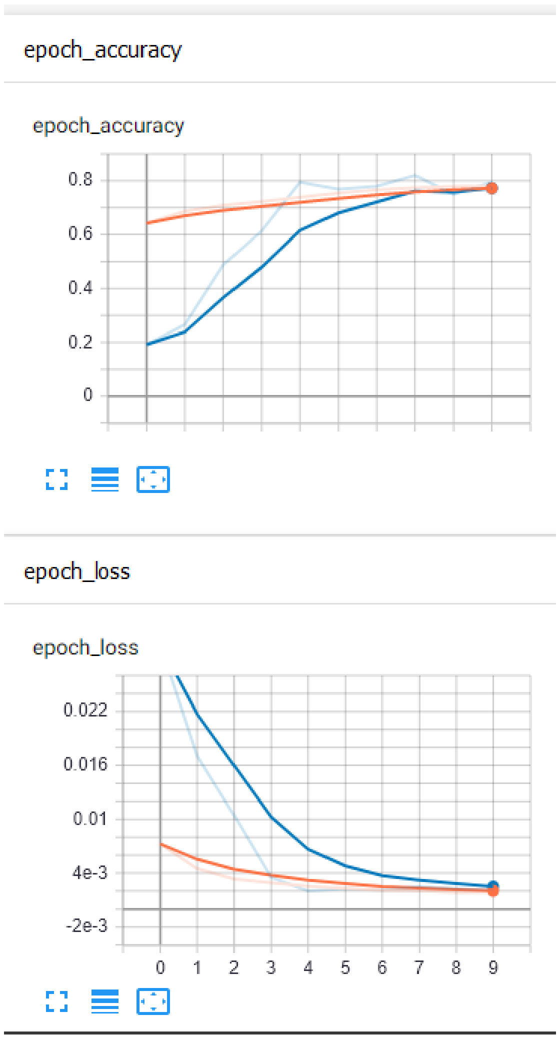


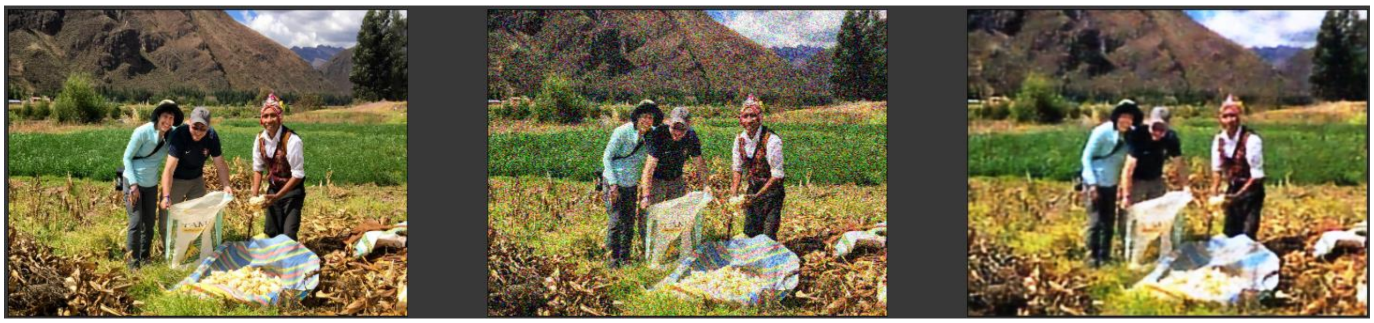


Original **Noisy** **Deconstructed**

The results are bad. The SGD took a very slow approach and did not reach a good result.

Run 4:





Original **Noisy** **Deconstructed**

The results are good. The change in loss function reduced accuracy a bit. But the Training and Validation accuracy are way closer than the first ADAM/MSE.

7. Possible strategies for improvement and conclusion

Possible strategies for improvement would be a lot finer tuning with the hidden layers with ADAM and MSE. Additionally, the dataset should initially have more images instead of the simple number of 500. The Dataset Augmentation can sometimes be inefficient. Also, more GPU and RAM power is required.

To conclude, the ADAM and RMSPROP optimizer with the MSE loss function have proven to be the most optimal in testing. Additionally, the SGD/NAG optimizer has proven to be least effective due to it being more effective on very shallow networks.

References:

[1] L. Gondara, “Medical image denoising using convolutional denoising autoencoders,” *arXiv.org*, 18-Sep-2016. [Online]. Available: https://arxiv.org/abs/1608.04667. [Accessed: 27-Apr-2020].

[2] P. Arbelaez, “Contour Detection and Image SegmentationResources,” *UC Berkeley Computer Vision Group - Contour Detection and Image Segmentation - Resources*, 2011. [Online]. Available: https://www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html. [Accessed: 02-May-2020].

[3] A. Dertat, "Applied Deep Learning - Part 3: Autoencoders", *Towards Data Science*, 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798. [Accessed: 14- Mar- 2020].

[4] "Unsupervised Feature Learning and Deep Learning Tutorial", *Ufldl.stanford.edu*, 2020. [Online]. Available: http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/. [Accessed: 14- Mar- 2020].