Analyzing Optimization of Deep Representation Learning QoS on TPU with TensorFlow

[1]T. Tritva J Kiran, [2]Dr. Pramod Pandurang Jadhav
[1]Ph.D. Scholar, CSE Department, Dr. A.P.J. Abdul Kalam University, Indore, MP, India.
[2]Professor, CSE Department, Dr. A.P.J. Abdul Kalam university, Indore, INDIA
{tritvajkiran@gmail.com, ppjadhav21@gmail.com}

ABSTRACT:
Auto encoders are a type of unsupervised learning technique of artificial neural networks that are mainly used to perform a task of data encoding or we call Representation learning. Representation Learning known as Auto Encoders are the extension part of Deep Learning which uses a kind of reverse engineering with basic kind of wheel architectures. auto encoders basically work by applying a dimensionality reduction that's very similar to principal component analysis or PCH. and PCH simply tries to perform dimensionality reduction. This work focused on the Quality of Service (QoS) in terms of Accuracy of the Deep Representation Learning Model optimization analysis which is known as Auto Encoders of Deep Learning model. Auto Encoder model QoS is tested on TPU of Intel® Core™ i3-7100U CPU using TensorFlow 2.0. Results are encouraged.

Keywords: Representation Learning, Auto Encoders, Deep Learning, TensorFlow, TPU, Artificial Neural Network, Unsupervised Training, QoS.
DOI Number: 10.48047/NQ.2022.20.21.NQ99087 NeuroQuantology2022;20(21): 723-732

I. INTRODUCTION

[1][3][5] Autoencoders are an unsupervised learning model in where we have an effect on neural networks for the undertaking of Representation learning. [4][5] Explicitly, we are going to predict a neural network structure such that we put in enforce a barrier withinside the structure which forces a flattened data representation of the novel input. [1][5] If the input parameter attribute were each sovereign of each other, this compression and succeeding modernization would be a very problematic task. [1][5] However, if some type of creation ensues in the data (i.e., associations between input features), this structure can be learned and accordingly leveraged when manufacture the input complete the network's block.

[1][3][5] Autoencoders are a category of artificial neural networks which are mostly used to execute a mission of data encoding a representation learning.

Applications of autoencoders include:

- Anomaly detection
- Data denoising (ex. images, audio)
- Image inpainting
- Information retrieval

[1][3][5] for this contemporary effort take an unlabelled dataset and mount it as a supervised learning problem worked with outputting $\hat{x}$, a renewal of the first input x. [1][5] This network can be trained by minimizing the reconstruction error, L $(x, \hat{x})$, which measures the differences between our original input and the consequent reconstruction. [1][5] The barrier is a significant feature of our network design; without the presence of a data block, our network could easily learn to simply memorize the input values by passing these values along through the network. [1][5] A barrier constrains the amount of data that can pass through entire network, making a learned compression of the input data.

[1][5] The autoencoder prototypical consistencies the following: Searching to the inputs adequate to accurately build a restructuring. [1][5] Unresponsive adequate to the inputs that the model doesn't basically learn or overfit the training data. [1][5] This trade-off navies the model to preserve only the variations in the data obligatory to restructure the input without holding on to severances within the input. [1][2][5] For maximum cases, this comprises building a loss function where one term reassures our model to be subtle to the inputs like reform loss L $(x, \hat{\mathbf{x}})$ and a second term discourages memorization/overfitting means an additional regularizer.

L $(x, \hat{x})$ + *regularizer*

[1][5] Naturally augment a scaling constraint in front of the regularization term so that we can fine-tune the trade-off between the two objectives.

[1][2][3][5] The modest architecture shown in figure (1) for making an autoencoder is to constrain the numeral of nodes present within the hidden layer(s) of the network, limiting the amount of information that can flow through the network. [1][5] By fining the network rendering to the reconstruction error, our model can acquire the most important features of the input data and how to unsurpassed reconstruct the novel input from an encoded state. [1][5] Preferably, this encoding will study and label hidden features of the input data.

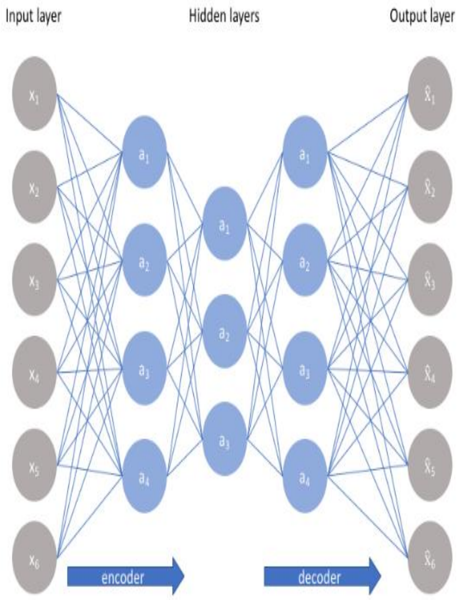

Figure (1): Modest Auto Encoder

[3] Neural networks are accomplished with learning nonlinear relationships, the thought of more powerful nonlinear generality. Autoencoders are expert of data nonlinear manifolds (a manifold is defined in modest terms as a continuous, non-intersecting surface). The variance between these two styles is visualized below figure (2).

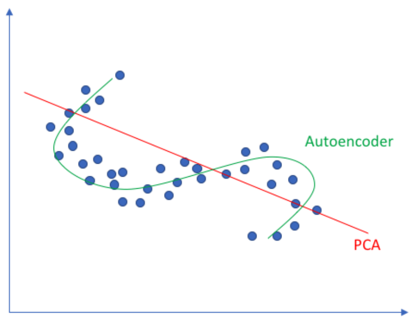

Figure (2): Linear vs Nonlinear dimensionality reduction

[1][3][5] For higher dimensional data, autoencoders are accomplished of learning a

compound symbol of the data (manifold shown in figure (3)) which can be used to describe observations in a lower dimensionality and congruently decoded into the novel input .

[1][5] Partial autoencoder has no clear regulation term - basically train the model translation to the restructuring loss. [1][5] Hence, the only way to confirm that the model isn't memorizing the input data is the confirm that we've adequately constrained the number of nodes in the hidden layer(s).

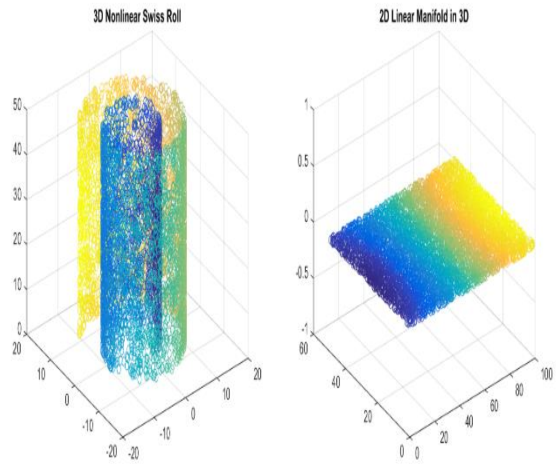

Figure (3): manifold

[1][2][3][5] For deep autoencoders, we must also be conscious of the volume of our encoder and decoder models. [1][5] Even if the barrier layer is only one hidden node, it's still likely for our model to learn the training data provided that the encoder and decoder models have passable fitness to learn some random function which can map the data to list. [1][5] Given the fact that our model to learn hidden features within our data, and our autoencoder model is not simply learning an efficient way to memorize the training data. [1][5] Similar to supervised learning problems, we can employ various forms of regularization to the network in order to reassure upright generalization properties; these techniques are discussed below.

A. Sparse autoencoders:

[1][5] Sparse autoencoders offer us another method for presenting a statistics barrier without requiring a reduction in the number of nodes at our hidden layers. [1][5] Then construct our loss function such that we fine activations within a layer. [1][5] For any certain observation, we'll boost our system to acquire an encoding and decoding which only relies on activating a trivial number of neurons. [1][5] It is noting that this is a different method to regularization, as we generally regularize the weights of a network, not the activations [3].

[1][5] A general sparse autoencoder is pictured below figure (4) where the denseness of a node agrees with the level of activation. [1][5] It specifying that definite nodes of a trained model which activate are data-dependent, diverse inputs will result in activations of different nodes through the network.

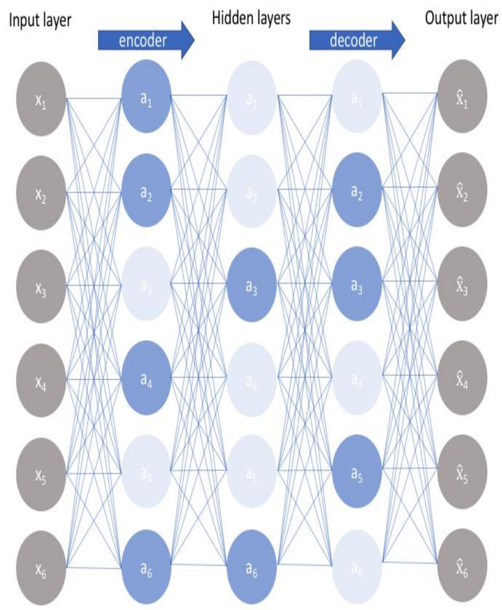

Figure (4): Sparse autoencoder

[1][5] One outcome of this statistic is that we allow our network to alert individual hidden layer nodes to specific attributes of the input data. [1][5] However, an undercomplete autoencoder will use the whole grid for all observation, a sparse autoencoder will be

prescribed selectively activate areas of the network reliant on the input data. [1][5] As a result, limit the network's capacity to learn the input data without limiting the networks capability to extract features from the data. [1][5] This allows us to study the latent state representation and regularization of the network discretely, such that we can take a latent state representation (i.e., encoding dimensionality) in accord with what changes intellect given the framework of the data while imposing regularization by the sparsity constraint.

[1][5] There are two ways to apply thinness constraint; both involve measuring the hidden layer activations for each training set and adding some term to the loss function in order to fine extreme activations.

[1][2][4][5] These are: **L1 Regularization:** We can add a term to our loss function that fines the complete value of the vector of activations *a* in layer *h* for observation *i*, scaled by a regulation parameter λ.

$$\mathcal{L}\left(x,\hat{x}\right)+\lambda\sum_{i}\left|a_{i}^{(h)}\right|$$

[1][5] KL-Divergence: In kernel, KL-divergence is a measure of the difference between two probability distributions. The sparsity constraint ρ which indicates the average activation of a neuron over a collection of samples. [1][5] This expectation can be calculated as

ρ^j=1m∑i[a(h)I (x)] ρ^j=1m∑i[ai(h)(x)]

where the subscript j denotes the specific neuron in layer *h,* summing the activations for mm training observations denoted individually as *x*. [1][5] In core, by confining the average stimulation of a neuron over a collection of samples we're encouraging neurons to only fire for a subset of the annotations. [1][5] We can define ρ as a Bernoulli random variable distribution such that we can control the KL divergence to compare the ideal distribution ρ to the observed distributions over all hidden layer nodes ρ^j.

L (x, x^) +∑jKL(ρ||ρ^j)

[1][5] The Bernoulli distribution resembles relatively fit with creating the probability a neuron will fire.

[1][5] The expression for the KL divergence between two Bernoulli distributions is given below:

∑j=1l(h)ρlogρρ^j+(1−ρ) log1−ρ1−ρ^j∑j=1l(h)ρlogρρ^j+(1−ρ) log1−ρ1−ρ^j.

This loss function is shown below figure (5) for an ideal distribution of ρ=0.2ρ=0.2, consistent with the minimum (zero) penalty at this point.

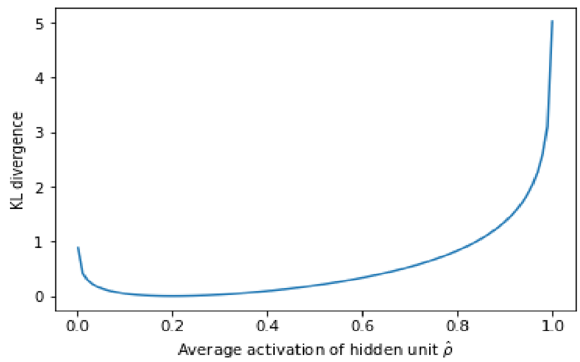

Figure (5): Loss function

B. Denoising autoencoders:

[1][2][5] Above concept of training a neural network where the input and outputs are equal and our model task is replicating the input as closely as probable while passing through some sort of data fence. [1][5] Additional method to developing a generalizable model is to somewhat corrupt the input data but still continue the unaffected data as our target output.

[1][5] With this method shown in figure (6), our model isn't able to simply develop a mapping which learns the training data because our input and target output are no longer the same. [1][5] Relatively, the model

learns a vector field for mapping the input data to a lower-dimensional manifold that a manifold calls the high-

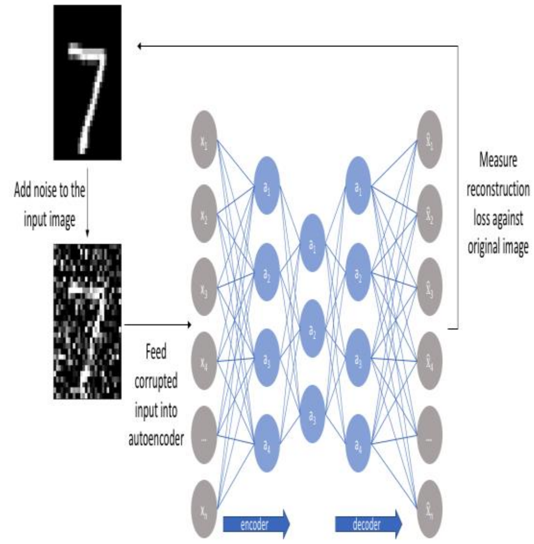

Figure (6): mapping

density area where the input data focusses, if this manifold exactly defines the expected data, we've successfully cancelled out the added noise.

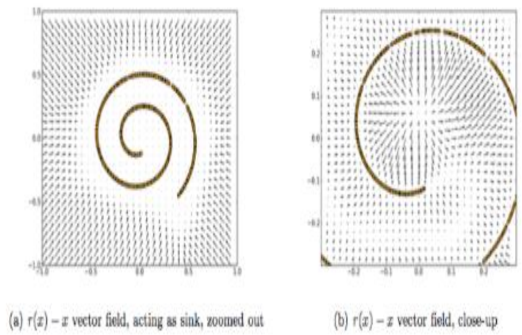

Figure (7): vector field

[1][5] The above figure (7) showing the vector field defined by comparing the reconstruction of xx with the original value of xx. [1][5] The points denote training samples prior to the addition of noise. See, the model has learned to regulate the corrupted input to the learned manifold. [1][5] This vector field is characteristically well behaved in the areas where the model has observed through training. [1][5] In natural data distribution, the transformation error is both large and does not always point in the direction of the true distribution as shown in figure (8).

[1][5] Contractive autoencoders: we can guess that for very alike inputs, the learned encoding would also be very similar. We can explicitly train our model for the derivative of the hidden layer activations is insignificant with respect to the input. [1][5] For small changes to the input, we should maintain a very analogous encoded state. [1][5] It is analogous to a

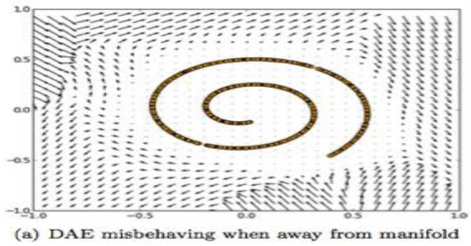

Figure (8): DAE misbehaving when away from manifold

denoising autoencoder and those trivial perturbations to the input are principally considered noise and that we would like our model to be robust against noise. [1][5] It is prominence of denoising autoencoders make the reform function (i.e., decoder) resist small but finite-sized agitations of the input, while contractive autoencoders make the feature taking out method (i.e., encoder) face up to minuscule agitations of the input.

[1][5] Because we're explicitly encouraging our model to learn an encoding in which similar inputs have similar encodings, we're essentially forcing the model to learn how to contract a neighbourhood of inputs into a smaller neighbourhood of outputs. [1][5] Point out how the slope (i.e., derivative) of the reconstructed data is basically zero for local neighbourhoods of input data depicted in figure (9)

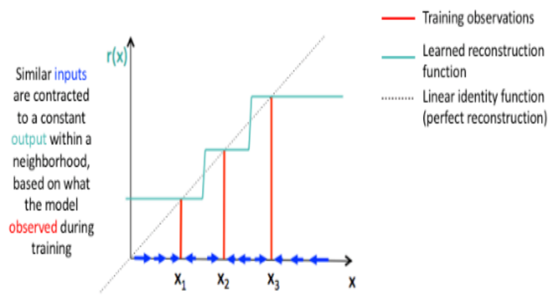

Figure (9): learning during training method

[1][5] Achieve this by making a loss function which penalizes large derivatives of our hidden layer activations with respect to the input training examples, essentially penalizing instances where a insignificant variation in the input primes to a large change in the encoding space.

[1][2][3][5] In fancier mathematical terms, we can craft our regularization loss term as the squared Frobenius norm ‖A‖F‖A‖F of the Jacobian matrix JJ for the hidden layer activations with respect to the input observations. [1][5] A Frobenius model is basically an L2 model for a matrix and the Jacobian matrix simply represents all first-order partial derivatives of a vector-valued function (in this case, we have a vector of training examples).

[1][5] For mm observations and nn hidden layer nodes, these values we compute using below expression.

$$\mathbf{J} = \begin{bmatrix} \frac{\delta a_1^{(h)}(x)}{\delta x_1} & \cdots & \frac{\delta a_1^{(h)}(x)}{\delta x_m} \\ \vdots & \ddots & \vdots \\ \frac{\delta a_n^{(h)}(x)}{\delta x_1} & \cdots & \frac{\delta a_n^{(h)}(x)}{\delta x_m} \end{bmatrix}$$

[1][5] Concisely, we can express our complete loss function as

L (x, x^) + λ∑i‖‖∇xa(h)i(x)‖‖2

where ∇xa(h)i(x)∇xai(h)(x) defines the gradient field of our hidden layer activations with respect to the input xx, summed over all ii training examples.

II. IMPLEMENTATION

The overall idea of auto encoders is that feed the exact same input data as an input and feed it as an output as well.

First take x3 and x test and perform some data normalization. Then to create some noise with the same dimensions and the original data and feed that as an input to auto encoder. Creating basically random noise with the size of 28 by 28. Add the noise factor. It should create my simple image noisy.
Do that for the entire training dataset. Which is 60000 images, and to confirm the dimensions of convolution a neural network that built before which is simply our auto encoder network which is consist of an encoder and decoder network. Now, we have a bunch of convolutions Max pooling. Then here use up sampling and convert to transpose. Now we have been able to compile our model and fit our model as well. First, let's confirm the dimensions. So, first if we have as trade equals to 1 and we use we'll call it same padding and same pattern means that the output would be equal to the inputs the dimensions of the output would be equal to the input.

The output dimension equals to the input image dimension W minus the F which is the size of kernel
or filter plus two P, P stands for how many zero patterns we have around the image divided by S, which is stride. Shifting kernel on top of image then add one to it and go, to know how P is calculated, P would be equal to whatever kernel size is which is F minus 1 divided by 2.

In this work, I used a filter that's 3 by 3. So, it's 3 minus 1 divided by 2 would give me one that means I need to add zeros here around the image just like padding of zeros subsequently the output would be identical to the input. The output would be equal executive same as the input because I need padding.

Let's assume that I want to build in one layer a single layer and this layer is a convolution layer. It consists of 16 filters and each filter a 3 by 3 stride of 1 padding same and input dimensions is 28 by 28 by 1. If we run that will be the output shape. Applied the convolution to your image and that's what you get. So, here we have our W which is dimension of my image 28 by 28. So, w is 28. My filter size is actually 3. My s which is my stride. In this work chose start equal to 1 and the padding will be equals P equals 2 F minus 1 divided by two equals two. If you substitute F in this case with 3 so, 3 minus one that's 2 divided by two will be equal to one. So, you need to add just like one pixel just a layer of pixels which contain zero. So, p equals to 1. Now I can substitute in the equation so W minus effortless 2 P developed by s plus 1, 28 minus 3 plus two times one divided by one plus one that will give me 28. So, the output is 28 by 28. These are my future maps and the depth of the end of all of them basic 16 and the total number of parameters will be 160 in this case.

We need to control the dimension of the image, the filter, stride, and the padding. These four parameters will determine the output dimension and that's crucial especially when it comes to auto encoders because we need the output to have dimensions of 28 by 28 by one. Now, apply basic classification mess around with it. Here are the dimensions have to match right. Now the predict method to train autoencoder is passing along X test noisy which basically comprise all the noisy data the testing data. The testing data is the subset of data that the model has never seen before during training. Now run the auto encoder to compile it. Now, apply the fit method to it. Run for one hundred epics. Now apply the predict method to it.

Now my output is 15 images each is 28 by 28 by one makes sense. Now my auto encoder basically took a sample of images that it has never seen in the past. So, here the data all of them are noisy. And now the auto encoder has been able to remove all the noise

III. RESULTS

With My Random factor = (28, 28) and Noise point = 0.2 the result I got from my model is depicted in below figure (10).

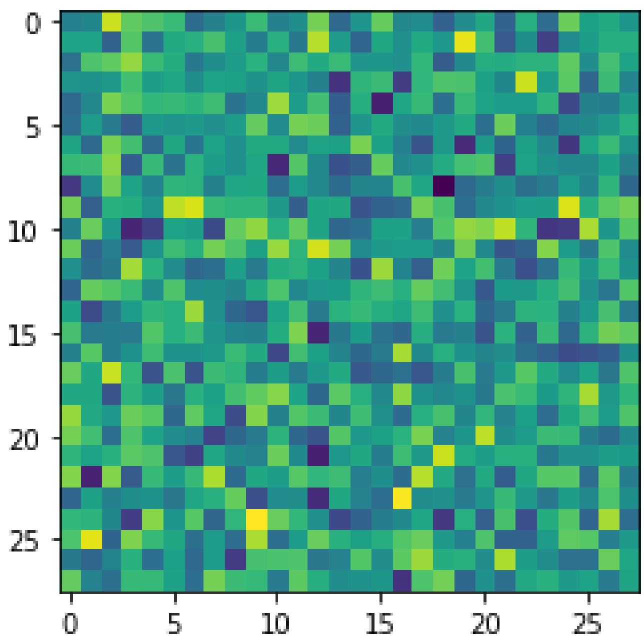

Figure (10): Noise to pad

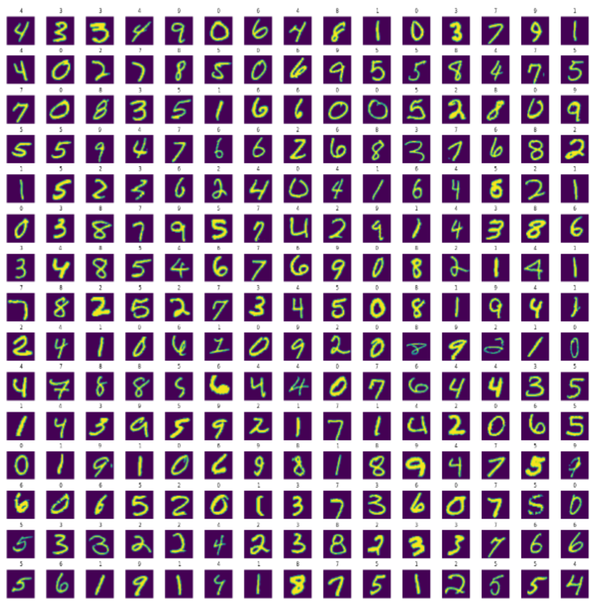

Figure (11): noise to pad on my dataset

By applying above noise, the result of dataset is resulted with noise is depicted in figure (11) and individual noisy numbered I extracted to depict as shown in figure (12) and figure (13).

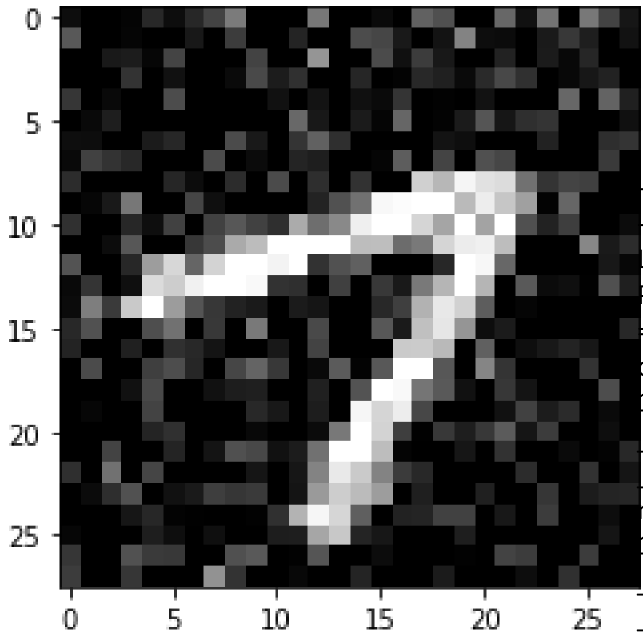

Figure (12): noise Output = (60000, 28, 28)

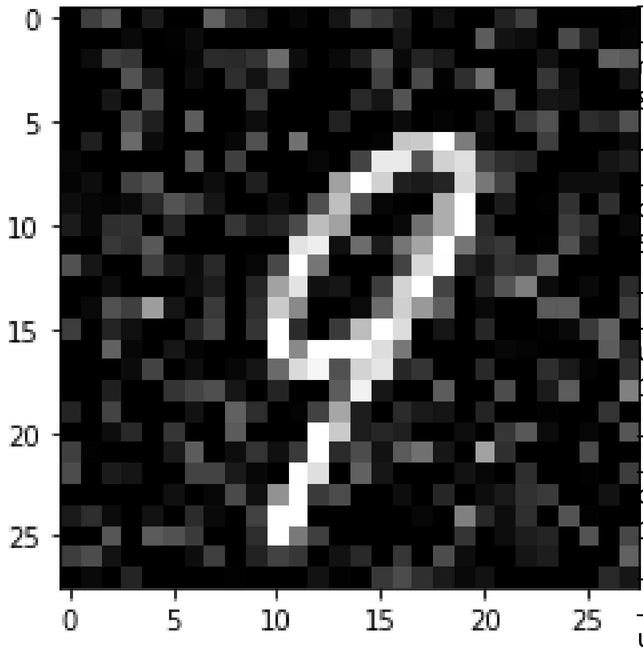

Figure (13): noisy output

All the parameters applied on each layer are clearly shown below during the training of my model and also given all layers parameter values for 4 sequential models.

```
--------------------------------------------------------------
-
Layer (type)                 Output Shape
Param #
=====================================
conv2d (Conv2D)              (None, 28, 28, 16)
160
_____________________________________________
____
max_pooling2d (MaxPooling2D) (None, 14,
14, 16)      0
_____________________________________________
____
conv2d_1 (Conv2D)            (None, 14, 14, 8)
1160
_____________________________________________
____
max_pooling2d_1 (MaxPooling2 (None, 7, 7,
8)        0
_____________________________________________
____
conv2d_2 (Conv2D)            (None, 7, 7, 8)
584
_____________________________________________
____
up_sampling2d (UpSampling2D) (None, 14,
14, 8)       0
_____________________________________________
____
conv2d_transpose (Conv2DTran (None, 14,
14, 8)       584
_____________________________________________
____
up_sampling2d_1 (UpSampling2 (None, 28,
28, 8)       0
_____________________________________________
____
```

conv2d_transpose_1 (Conv2DTr (None, 28, 28, 1) 73
=====================================
Total params: 2,561
Trainable params: 2,561
Non-trainable params: 0

-

Model: sequential_1
____

Layer (type) Output Shape Param #
=====================================
conv2d_3 (Conv2D) (None, 28, 28, 16) 160
=====================================
Total params: 160
Trainable params: 160
Non-trainable params: 0

Model: sequential_2
____

Layer (type) Output Shape Param #
=====================================
conv2d_4 (Conv2D) (None, 14, 14, 16) 160
=====================================
Total params: 160
Trainable params: 160
Non-trainable params: 0

-

Model: sequential_3
____

Layer (type) Output Shape Param #
=====================================
conv2d_5 (Conv2D) (None, 26, 26, 16) 160
=====================================
Total params: 160
Trainable params: 160
Non-trainable params: 0

-

Model: sequential_4
____

Layer (type) Output Shape Param #
=====================================
conv2d_6 (Conv2D) (None, 13, 13, 16) 160
=====================================
Total params: 160
Trainable params: 160
Non-trainable params: 0

-

The final output is depicted in figure (14).

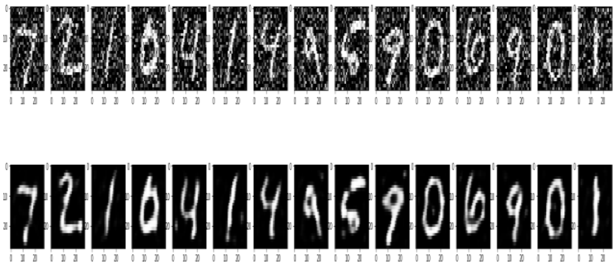

Figure (14): final output

IV. CONCLUSION

The Deep Representation Learning is a neural network structure proficient of learning construction confidential data in order to develop a flattened depiction of the input. Numerous assorted modifications of the general autoencoder architecture are with the goal of confirming that the compressed model signifies expressive attributes of the original data input; the key task is when working with autoencoders which receiving your model to basically learn expressive and generalizable latent space representation.

As autoencoders learn how to compress the data constructed on attributes exposed from data through training, these models are classically accomplished of reconstruction data corresponding to the class of observations of which the model observed all through training.

In this work the Deep Auto Encoder model QoS is tested on TPU of Intel® Core™ i3-7100U CPU using TensorFlow 2.0. Results are encouraged.

Future Work: I would plan to extend my work to testing on HPC using TPU of TensorFlow by extending intermediately parameters by increasing middle layers for different structural architectures of deep learning convolutional neural network.

V. REFERENCES

[1] Guillaume Alain and Yoshua Bengio, What Regularized Auto-Encoders Learn from the Data Generating Distribution, arXiv, 19 aug 2014.

[2] Pascal Vincent, Hugo Larochelle, Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion, Journal of Machine Learning Research 11 (2010) 3371-3408

[3] G. E. Hinton, R. R. S. dinov, Reducing the Dimensionality of Data with Neural Networks, Science 28 Jul 2006.

[4] Peng Liu, Peijun Zheng and Ziyu Chen, Deep Learning with Stacked Denoising Auto-Encoder for Short-Term Electric Load Forecasting, energies, MDPI, 25 June 2019.

[5] https://www.jeremyjordan.me

[1]Author:

I am Mrs. T. Tritva Jyothi Kiran with 10 years of work as Assistant Professor in Computer Science Department in various Engineering Colleges. Currently I am working in AKNU. Previously I have completed AICTE funding Project on IEEE802.11e in JNTUH and published in IEEE conference. I have been awarded Two times for the National Award for Excellence Adarsh Vidya Saraswathi Rastriya Puraskar from Glacier Global Management in 2020. And Received Women Researcher Award from 9th international conference organized by VDGOOD Professional Association. Extant I am working in the research domain Deep Learning using TensorFlow and Swarm Intelligence. You can find my Lectures during COVID in my Blog is tritvajyothikiran.blogspot.com and in Tritva Jyothi YouTube Channel.

[2]Author:

I am Dr. Pramod Pandurang Jadhav, Professor in Dr. A.P.J. Abdul Kalam University, Indore, India. My research area of interest on software engineering Networking, IOT, Distributed system, security, Data science, Artificial Intelligence, Machine learning. I am a life time member of professional Body called ISTE (**I**ndian **S**ociety for **T**echnical **E**ducation). And awarded for Best Professional Award from ESN International Publication Chennai. I published various paper in Standard International Journal and Conference like Scopus, SCI etc, and guided various Under graduate, Post graduate Engineering student and PhD Candidate as a Research Guide. I have more than fourteen years of teaching experience in Pune University.