Study and Implementation of Auto-encoders for Generating Sketches from Face Images and Faces from Sketch Images.

Prathamesh Kulkarni (Student ID: 36004026)
Dept. of Computer Science and Communication
Lancaster University
Lancaster, England
p.p.kulkarni@lancaster.ac.uk

***Abstract*—The aim of the paper is to use autoencoders to generate sketches from a given face images and an image from a given sketch. To make this possible various autoencoder are discussed and implemented to see how each of them perform and if they can generate quality images. Deep autoencoders, convolution autoencoders and ensemble autoencoder are implemented on CUHK Face Sketch Database to check the performance. It was found that the ensemble autoencoders performed better than deep dense and convolution autoencoders.**

***Index Terms*—Auto-Encoders, Dense Auto-Encoders, Convolution Auto-Encoders, Ensemble Auto-Encoders.**

I. Introduction

Deep learning has always been challenging when it comes to generating images. It is because the amount of data is so large, the training phase requires a great deal of time and resources, and to gain high-quality images, a lot of optimization is required. But deep learning research has evolved significantly in recent years. Using the latest techniques, it became possible to interpret more complicated AI tasks. Various techniques like image generation, data denoising, dimensionality reduction, image compression, feature extraction, etc., have gained popularity due to the accuracy with which they perform. To generate images, autoencoders are used. As unsupervised learning algorithms, autoencoders play an integral part in deep learning architectures. An encoder and a decoder are the elementary elements of autoencoders. During the learning process, autoencoders use images from the input, which can then be reconstructed at the output [3], [4]. Autoencoders are used in this paper to generate sketches of people from their face images, as well as to generate coloured images from those sketches. The purpose of this project is to see which autoencoders can handle the task effectively and whether combining these autoencoders would result in more accurate models. The implications of this are huge. The authorities can use it by feeding in sketches of suspects so that the models will show what the suspect actually looks like. First, the paper explains how the data were processed. Next, the autoencoders are explained. Models such as deep autoencoders, convolutional autoencoders, and ensemble autoencoders are described and implemented. Following that, each of these models is compared in order to see how it performed and if the images were correctly generated.

II. Literature Review

In the past, autoencoders have been a subject of discussion in a variety of fields. A good example is anomaly detection in [6] where autoencoders were used to detect anomalous sounds. It is based on the fact that, if data were supplied other than the conditions provided during training, it would be difficult to reconstruct the latent attributes, since they would not be adapted for the sound that was never heard. Consequently, the outlier sound gives off very high reconstruction losses, which can be easily identified by a proper threshold as an anomaly. A convolutional and dense autoencoder was used to achieve this. In [9], dimensionality reduction of images is discussed as a means of improving reinforcement learning models. A lower dimensionality of the images is crucial as it makes it easier for the reinforcement models to learn from them. Deep autoencoders can be used for this purpose, as discussed in the paper. Feature extraction is one of the tasks in converting faces to sketches, discussed in [22]. During the reconstruction process, hidden features present in the input data are learned and the error is reduced. As features are encoded, a distinct set of combinations is generated. According to [17], autoencoders are useful in the generation of images. The system generates images similar to input images, such as faces or scenery. It is possible to convert any black-and-white image into a coloured image with these autoencoders. It is possible to determine the colour from what is in the picture. In this paper, the goal is to convert images of faces into sketches and sketches into images of faces by colouring them. Various applications are used to achieve this, such as dimension reduction, feature extraction, and image generation. In the [16] reference, neural networks were grouped together to create a better performing model. As part of this paper, the deep dense autoencoder and convolutional autoencoder are combined to create a model which is better at converting faces into sketches and vice versa.

III. Methodology

A. Preprocessing

As the models only work with numerical data, it is necessary to provide the image data in a proper format before training them. Preprocessing begins with setting the correct working directory for importing the images. It is important to specify the exact directories where the images and sketches can be found. As the images and sketches are all associated with one another, it is necessary to assign them a proper order. The reason for this is, if not done correctly, there will be a label mismatch, which means the model will be trained on data with a different label. This will severely hamper the performance of the model. After arranging the files correctly, it is necessary to resize and assign colour channels to the images and sketches. Taking in images of the same size and corresponding colour channel is crucial, meaning that if they are coloured, the images will be made up of red, green, and blue colours, so the number 3 will be assigned to the image. The final image has the shape 256x256x3, where 256 by 256 is the image's size and 3 is the RGB colour. The number of images in the dataset is very small; therefore, using different augmentation methods can increase the size of the training data. For these applications, more training images mean better models. The images are rotated and flipped in varied ways, and then added to the original set of data. These images are converted to arrays, which then convert the images to a numerical format and are stored. Because these images are fed to the model one at a time, the images are again reshaped to 1x256x256x3, where the "1" denotes only one image. Thus, the total number of images obtained is 1408 which includes both coloured and sketched images. Having obtained all the image arrays, training and testing data sets need to be created. The ratio of training to testing is 90:10, meaning 90 percent of the images are used for training, while 10 percent are used for testing.

B. Auto Encoders

With an artificial neural network, an autoencoder can learn data encodings unsupervised. Each autoencoder includes an encoder and a decoder. Encoders are used to compress input data, while decoders are used to decompress it. Autoencoders can learn a low-dimensional representation of high-dimensional images with an autoencoder by training the network to capture the most important parts of input images. Hence, an autoencoder that can reproduce its input should in principle have a latent space that encodes all the necessary information for representing the original data, while at the same time being smaller than the input. [1]

Many applications can be accomplished with autoencoders, among them denoising, image compression, and the generation of the image data. In this paper, the autoencoders are used to generate images. To be specific the autoencoders extract the information from the images consisting of faces of people and convert them into sketches and vice versa.

Figure 1 illustrates the basic principles of autoencoding. On the left side are the inputs. A high-dimensional input is given here and reduced to a lower-dimensional output in the hidden layer. However, the important aspects of the data are preserved. Data is being regenerated on the decoder side. There are many neurons in each layer in practice, and there can also be many hidden layers.

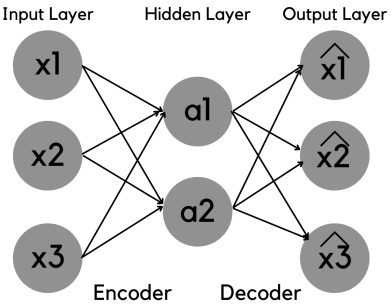

Fig. 1. A simple autoencoder

Deep Auto Encoder

Those autoencoders which have more than one pair of layers are referred to as deep. A deep autoencoder typically consists of two deep belief networks with four or five shallow layers. These shallow layers represent the encoding and decoding portions of the network. A good strategy for improving the learning power of an autoencoder is to stack its layers on top of each other. As a result, deep autoencoders are considered deep belief networks, since they contain latent variables that are visible only to single layers in the forward direction. The number of layers is usually higher than autoencoders that have a pair of layers. [8] The image below illustrates an implementation of one of these models. An input layer is shown, followed by a sequential model, which is an encoder consisting of hidden layers made up of 64, 32, 16 neurons. The decoder, therefore, has the reverse sequence of layers with 16, 32, 64 neurons and an output layer.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 16)                12585584
_________________________________________________________________
sequential_1 (Sequential)    (None, 256, 256, 3)       12782176
=================================================================
Total params: 25,367,760
Trainable params: 25,367,760
Non-trainable params: 0
_________________________________________________________________
```

Fig. 2. Model 1 Summary

Convolution Auto Encoders

Convolutional neural networks are widely used for the modeling of image data. It can provide a more comprehensive model since it retains the connections between each pixel.

A convolution autoencoder uses convolution networks rather than deep neural networks. Deep autoencoders are feedforward networks that contain encoders and decoders. The encoder uses convolutional layers, while the decoder uses transposed convolutional layers instead of feedforward networks. Convolutions and pooling are used for the encoder, and deconvolution is used for the decoder. [2], [5]

A convolution autoencoder is shown in the figure 3 below. In order to derive a bottleneck representation of the image, several convolutions and maximum pooling operations are applied to the image.

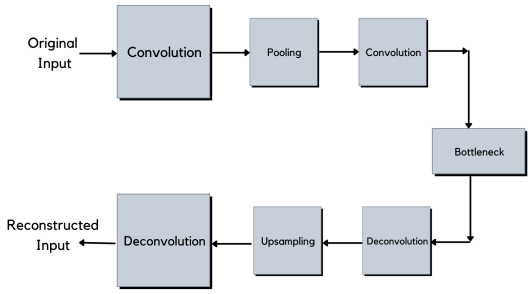

Fig. 3. The figure represents a convolution autoencoder.

Layer (type)	Output Shape	Param #
conv2d_26 (Conv2D)	(None, 256, 256, 64)	1792
max_pooling2d_6 (MaxPooling2	(None, 128, 128, 64)	0
conv2d_27 (Conv2D)	(None, 128, 128, 32)	18464
max_pooling2d_7 (MaxPooling2	(None, 64, 64, 32)	0
conv2d_28 (Conv2D)	(None, 64, 64, 16)	4624
max_pooling2d_8 (MaxPooling2	(None, 32, 32, 16)	0
conv2d_29 (Conv2D)	(None, 32, 32, 16)	2320
up_sampling2d_6 (UpSampling2	(None, 64, 64, 16)	0
conv2d_30 (Conv2D)	(None, 64, 64, 32)	4640
up_sampling2d_7 (UpSampling2	(None, 128, 128, 32)	0
conv2d_31 (Conv2D)	(None, 128, 128, 64)	18496
up_sampling2d_8 (UpSampling2	(None, 256, 256, 64)	0
conv2d_32 (Conv2D)	(None, 256, 256, 3)	1731

Total params: 52,067
Trainable params: 52,067
Non-trainable params: 0

Fig. 4. Model 2 Summary

A simple and an optimized convolution autoencoder are discussed in the paper. The first model consists of a convolution autoencoder with 5 layers consisting of 64, 32, 16, 32 and 64 neurons respectively. Figure 4 shows the model with seven convolution and three pooling layers, and three upscaling layers.

In the second model, neurons are arranged in increasing order on the encoder side and in decreasing order on the decoder side. The encoder is comprised of 6 layers with 16, 32, 64, 128, 256, and 512 neurons, respectively. On the decoder side, there are 6 layers comprising 512, 256, 128, 64, 32, and 16 neurons, respectively. In addition, batch normalization and dropout are implemented.

Ensemble Auto Encoders

An ensembling is the combination of multiple learning algorithms to improve existing models by combining several, thus producing a more accurate model. An ensemble model is shown in the figure 6.

Compared to traditional machine learning algorithms, deep learning algorithms have higher performance on their own, but they can be combined in a way to achieve even better performance. This is achieved by ensembling the previously implemented models in a bagging configuration to arrive at a final prediction. The reason behind implementing this is to see if the ensemble model will be able to generate high-quality images. [10] [11] [12] [13] [14] [15]

IV. Experiment Results and Discussion

Before understanding the results there are few key concepts that are necessary to understand how the training phase took place.

Batch Normalization

Data normalization involves transforming numerical data into one scale without distorting their shape. Deep learning or machine learning algorithm normally require the values to be converted to a balanced scale when given as an input. Normalizing partly ensures that our model will generalize correctly. By adding extra layers to deep neural networks, batch normalization makes neural networks faster and more stable. The new layer standardizes and normalizes the input it receives from a previous layer. Most neural networks are trained with a set of input data called a batch. Batch normalizations are also performed in batches, not as a single input. During the training process, batch normalization helps stabilize the distribution of internal activations. As a result of batch normalization, higher learning rates are also possible and initialization sensitivity is reduced. [18], [19]

Dropout

During training, neurons develop co-dependency between one another due to a fully connected layer, thereby curbing the individual power of each neuron. This results in an overfitting of the training data. In the training phase, ignoring neurons that are randomly selected is called dropout. This

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 256, 256, 3)]	0
sequential_4 (Sequential)	(None, 127, 127, 16)	768
sequential_5 (Sequential)	(None, 62, 62, 32)	8320
sequential_6 (Sequential)	(None, 30, 30, 64)	32768
sequential_7 (Sequential)	(None, 14, 14, 128)	131584
sequential_8 (Sequential)	(None, 6, 6, 256)	525312
sequential_9 (Sequential)	(None, 2, 2, 512)	2099200
sequential_10 (Sequential)	(None, 6, 6, 512)	4194304
sequential_11 (Sequential)	(None, 14, 14, 256)	2097152
sequential_12 (Sequential)	(None, 30, 30, 128)	524288
sequential_13 (Sequential)	(None, 62, 62, 64)	131072
sequential_14 (Sequential)	(None, 126, 126, 32)	32768
sequential_15 (Sequential)	(None, 254, 254, 16)	8192
conv2d_transpose_6 (Conv2DTr	(None, 255, 255, 8)	520
conv2d_transpose_7 (Conv2DTr	(None, 256, 256, 3)	99

Total params: 9,786,347
Trainable params: 9,784,491
Non-trainable params: 1,856

Fig. 5. Model 3 Summary

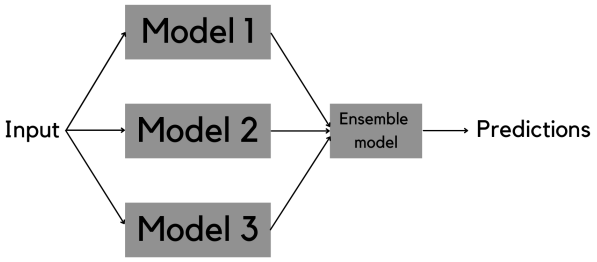

Fig. 6. An illustration of the ensemble model is shown.

means that certain neurons aren't taken into account while passing forward or backwards. A reduced network is left after each training stage by either dropping nodes or keeping them. Regularization of neural networks using dropout reduces the correlation among the neurons in the network, which allows for more efficient learning. Several types of layers

Model: "model_2"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[(None, 256, 256, 3)	0	
sequential_16 (Sequential)	(None, 256, 256, 3)	52067	input_3[0][0]
model_1 (Functional)	(None, 256, 256, 3)	9786347	input_3[0][0]
sequential_31 (Sequential)	(None, 256, 256, 3)	25367760	input_3[0][0]
add (Add)	(None, 256, 256, 3)	0	sequential_16[0][0] model_1[0][0] sequential_31[0][0]

Total params: 35,206,174
Trainable params: 35,204,318
Non-trainable params: 1,856

Fig. 7. Ensemble model summary

can be used with it, including dense fully connected layers, convolutional layers, and recurrent layers, such as the long-short-term memory network. [20] In the implementation of the third model dropout with 0.1 probability is used. This means 0.1 percent of the neurons from that layer will be dropped at random.

Adam Optimizer

Adam is an optimization algorithm for updating network weights iteratively based on training data, as opposed to the conventional stochastic gradient descent procedure. The name Adam comes from adaptive moment estimation. In the method, the first and second moments of gradients are used to compute individual adaptive learning rates. The benefits of both AdaGrad and RMSProp are included in Adam. Adam averages not only the first moment of the gradient but also the second moment. Adam is widely used in deep learning algorithms because it achieves accurate and fast results. [21]

Loss Functions

Mean squared error and mean absolute error were two loss functions that were used. The two loss functions given above were used because they produced better results. The mean squared error is calculated by squaring the difference between the predictions and the original predictions, then averaging it over the entire dataset. [25]

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(predicted_i - original_i)^2 \qquad (1)$$

Where N is the number of samples

Because the MSE gives greater weight to errors in the square part of the function, it is effective when it comes to ensuring that the model has no outlier predictions with huge errors.

In mean absolute error, an absolute value is applied to the difference between the predictions and the original predictions, and the mean of this value is then calculated over the entire set of data.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|predicted_i - original_i| \quad (2)$$

Where N is the number of samples

Unlike MSE, all errors will be weighted the same way since absolute values are taken.

It was found that convolution and ensemble models that used mean absolute error produced better images.

Structural Similarity

Structural Similarity Index (SSIM) is a metric for measuring how similar or dissimilar two images are. This metric compares an image to a reference image. It is possible to compare a generated sketch to a reference sketch to determine how similar they are. SSIM lies between -1 and 1 or 0 and 1. A score close to 1 indicates that images are highly similar to each other. Luminance, contrast, and structure all contribute to structural similarity. The small differences in noise and variation on individual pixels don't typically affect groups of pixels as much as they do on individual pixels in MSE, which compares the two images pixel by pixel. As the complexity of the models increased, so did the quality of the images generated resulting in a higher SSIM. [23]

Activation Functions

A neuron's Activation Function determines whether it should be activated or not. The Activation Function is responsible for generating output from the input values fed into a node. The back-propagation is possible because the gradients, as well as the error, are supplied to update the weights and biases. In this study, three non-linear functions of activation have been employed, namely Relu, Leaky Relu, and Sigmoid. [24]

1) ReLU
   The Rectified Linear Unit is referred to as ReLU. In ReLU functions, not all neurons fire simultaneously. In order for the neurons to be deactivated, the linear transformation's output must be less than 0. Zero is returned if the function receives a negative input, but it returns the value if the input is positive. Thus, it can give a range of outputs between 0 and infinity.
2) Leaky ReLU
   A leaky ReLU is an improvement over ReLU. Leaky ReLU has the same advantages as ReLU, plus it supports backpropagation, even for negative inputs.
3) Sigmoid
   It takes any real value as an input and gives back values in the range 0 to 1. Generally, a larger input (more positive) will result in a higher output value, while a smaller input (more negative) will result in a lower output value.

The first model, which is a deep autoencoder, was trained with a sigmoid activation function on the output layer and a relu activation function for each layer. The mean absolute error loss function was used with the Adam optimizer, and both accuracy and loss were assessed. Figure 8 demonstrate that even though the loss converges the model was not able to generate quality images. For the image to sketch conversion, accuracy was 33.4 percent with the SSIM of 0.79, and for the sketch to image conversion, the accuracy was 88.2 percent and SSIM 0.72.

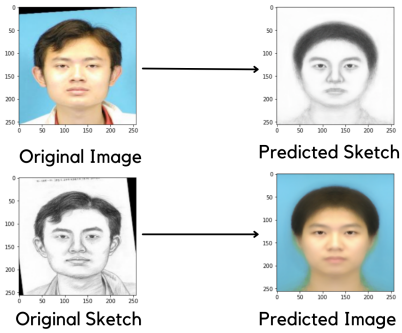

Fig. 8. Outputs from deep dense autoencoder.

We can see from the above results that model 1 was not able to regenerate quality images.

Second, we have a simple convolution autoencoder. Activation function here was relu, loss function was mean square error, and Adam optimizer was used. Compared with the mean absolute error loss function, the mean squared error provided better prediction. In face to sketch, the accuracy was 39.2 percent, with a SSIM score of 0.82. Conversely, sketch to face was 74.7 percent accurate with a SSIM score of 0.77.

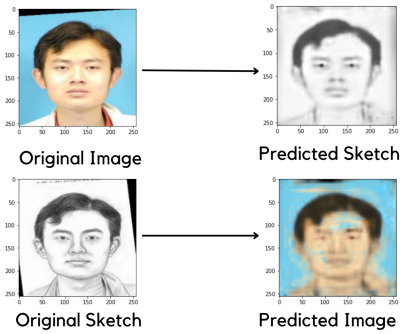

Fig. 9. Outputs from first convolution model (Model 2).

As shown above, Model 2 was better at predicting sketches, but not at generating faces based on sketches. This was an improvement over model 1.

Comparatively to model 2, the third model is a more complex convolution model. There are more layers in this model and it has features such as batch normalization and

dropouts. As activation function, leaky relu was used, while mean absolute error was used as the loss function, along with Adam optimizer with a learning rate of 0.001. In this model, the accuracy was 28.3 percent and the SSIM was 0.81 for face to sketch and 86.6 percent and 0.74 for sketch to face respectively.

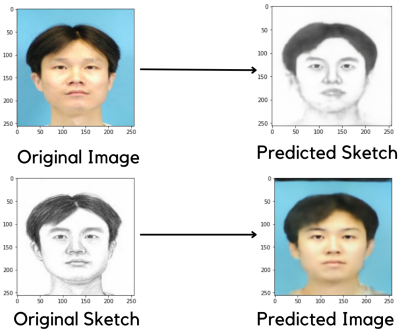

Fig. 10. Outputs from second convolution model (Model 3).

This model is much better able to predict both scenarios, as shown.

Finally, we have the ensemble of the above three models. All three models were added together in a bagging configuration. The loss function was mean absolute error and Adam was used as the optimizer. For face to sketch, the model gave an accuracy of 23 percent and SSIM of 0.78 and for sketch to face, it gave an accuracy of 90.1 percent and SSIM of 0.86.

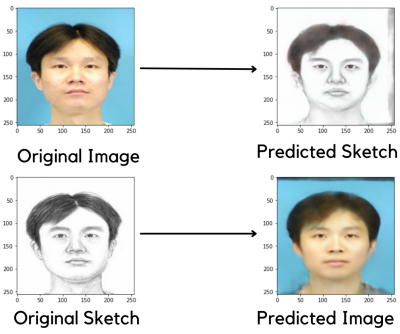

Fig. 11. Outputs from ensemble model (Model 4).

The sketches and faces predicted by this model are much better than those predicted by individual models, since it uses all models for prediction.

TABLE I
PERFORMANCE METRICS

Metric	Model 1	Model 2	Model 3	Model 4
Accuracy face2sketch	33.4	39.2	28.3	23
Accuracy sketch2face	88.2	74.7	86.6	90.1
SSIM face2sketch	0.79	0.82	0.81	0.78
SSIM sketch2face	0.72	0.77	0.74	0.86

It can be seen from table 1 that accuracy of all models for the sketch to face was was higher than face to sketch but that does not represent the model performance for generating images. For example, the ensemble model gave a poor accuracy for face to sketch yet when the images are compared visually they seem to be of better quality than other models. Hence SSIM was used to get a better idea and compare the images as it is a more relevant metric to judge the closeness of predictions to the original images. It can be observed that models 3 and 4 had higher SSIM scores as they were much more complex than models 1 and model 2. In terms of loss all the models showed a decreasing trend for the loss on both training and validation sets, eventually converging. But given the size of the data these results do not represent the actual generation capability of these models. This is because for deep learning tasks, specially if they are related to images, need a huge data which has a variety of images to actually learn useful features to properly predict for new images.

V. Conclusion

The aim of the paper was to implement autoencoders to convert face images to sketches and vice versa. This was accomplished successfully. As the complexity of models increased, the learning capability and the generation capability of the models also grew but this may not be the case in every situation. Since deep autoencoder is the most basic model, it performed poorly. Furthermore, the ensemble model, which was a combination of all three models, performed much better than individual models, as the ensemble model treated each model as a weak learner to create a stronger model and, ultimately, come up with a superior prediction of the images. Different loss functions and optimizers were examined to determine which performed best. This resulted in the two loss functions, namely mean square error and mean absolute error, performing better than other loss functions. The Adam optimizer outperformed other algorithms, such as stochastic gradient descent, RMSprop, and NAdam. While these models were able to produce the intended results, they could be adjusted a lot to get even better results, or different types of autoencoders or GANs could be more suitable for this task.

Acknowledgment

I would like to acknowledge:

1) Professor Dr. Richard Jiang For the lectures and the lab sessions and codes.
2) The creators of libraries such as numpy, pandas, tensorflow, keras and matplotlib.
3) Creators of the CUHK Face Sketch Database
4) Linda Marano for providing methods for preprocessing and covolution autoencoder. Code

References

[1] Iffat Zafar, Giounona Tzanidou, Richard Burton, Nimesh Patel, Leonardo Araujo, “Hands-On Convolutional Neural Networks with TensorFlow” Packt Publishing, August 2018.
[2] Nikhil Ketkar, Jojo Moolayil, “Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch”, Apress, April 2021.

[3] David Foster, “Generative Deep Learning”, O’Reilly Media, Inc., June 2019.

[4] Dr. Pablo Rivas, Laura Montoya, “Deep Learning for Beginners”, Packt Publishing, September 2020.

[5] Sudharsan Ravichandiran, “Hands-On Deep Learning Algorithms with Python”, Packt Publishing, July 2019.

[6] Alexandrine Ribeiro, Luıs Miguel Matos, Pedro Jose Pereira, Eduardo C. Nunes, Andre L. Ferreira, Paulo Cortez, Andre Pilastri, “Deep Dense And Convolutional Autoencoders For Unsupervised Anomaly Detection In Machine Condition Sounds”, Detection and Classification of Acoustic Scenes and Events 2020, 19 Jun 2020.

[7] Xiaolong He, Qizhi He, Jiun-Shyan Chen, “Deep autoencoders for physics-constrained data-driven nonlinear materials modeling”, Computer Methods in Applied Mechanics and Engineering, 1 November 2021.

[8] Alex Krizhevsky and Geoffrey E. Hinton, “Using Very Deep Autoencoders for Content-Based Image Retrieval”, University of Toronto - Department of Computer Science.

[9] Bharat Prakash, Mark Horton, Nicholas R. Waytowich, William David Hairston, Tim Oates, Tinoosh Mohsenin, “On the use of Deep Autoencoders for Efficient Embedded Reinforcement Learning”, arXiv:1903.10404v1, 25 Mar 2019.

[10] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici and Asaf Shabtai, “Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection”, Network and Distributed System Security Symposium, 27 May 2018.

[11] Henry W J Reeve, Gavin Brown, ”Modular Autoencoders for Ensemble Feature Extraction” JMLR: Workshop and Conference Proceedings 44 (2015).

[12] Jinghui Chen, Saket Sathe, Charu Aggarwal, Deepak Turaga, “Outlier Detection with Autoencoder Ensembles”, University of Virginia

[13] L. Yang, Y. Song, S. Gao, B. Xiao and A. Hu, ”Griffin: An Ensemble of AutoEncoders for Anomaly Traffic Detection in SDN,” GLOBECOM 2020 - 2020 IEEE Global Communications Conference, 2020, pp. 1-6, doi: 10.1109/GLOBECOM42002.2020.9322187.

[14] Hamed Sarvari, Carlotta Domeniconi, Bardh Prenkaj, Giovanni Stilo, ”Unsupervised Boosting-based Autoencoder Ensembles for Outlier Detection”, arXiv:1910.09754v1, 22 Oct 2019.

[15] Bingjun Guo, Lei Song, Taisheng Zheng, Haoran Liang, Hongfei Wang, ”Bagging deep autoencoders with dynamic threshold for semi-supervised anomaly detection,” Proc. SPIE 11321, 2019 International Conference on Image and Video Processing, and Artificial Intelligence, 113211Z (27 November 2019).

[16] Kyoungnam Ha, Sungzoon Cho, Douglas MacLachlan, ”Response models based on bagging neural networks”, Journal of Interactive Marketing, Volume 19, Issue 1, 2005.

[17] Panguluri, Koumudi and Kamarajugadda, Kishore, ”Image Generation using Variational Autoencoders”, IJITEE (International Journal of Information Technology and Electrical Engineering) 10.35940/ijitee.E2480.039520.

[18] Sergey Ioffe, Christian Szegedy, ”Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, arXiv:1502.03167, 2015.

[19] Sergey Ioffe, ”Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models”, arXiv:1702.03275, March 2017.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, ”Dropout: A Simple Way to Prevent Neural Networks from Overfitting”, Journal of Machine Learning Research 15 (2014) 1929-1958, June 2014.

[21] Diederik P. Kingma, Jimmy Ba, ”Adam: A Method For Stochastic Optimization”, Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[22] Marco Maggipinto, Chiara Masiero, Alessandro Beghi, Gian Antonio Susto, ”A Convolutional Autoencoder Approach for Feature Extraction in Virtual Metrology”, Procedia Manufacturing, Volume 17, 2018.

[23] Kieran G. Larkin, ”Structural Similarity Index SSIMplified”, Occasional Texts in the Pursuit of Clarity and Simplicity in Research. Series 1, Number 1, May 2015.

[24] Prajit Ramachandran, Barret Zoph, Quoc V. Le, ”Searching for Activation Functions”, arXiv:1710.05941, 27 Oct 2017.

[25] Alexei Botchkarev, ”Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology”, https://arxiv.org/ftp/arxiv/papers/1809/1809.03006.pdf.