Data compression and reconstruction based on Deep Neural Networks

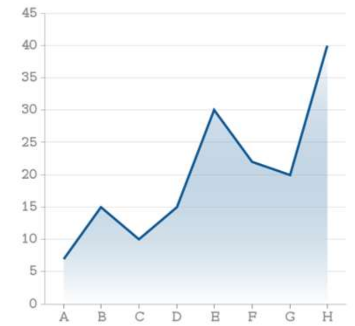
3A09

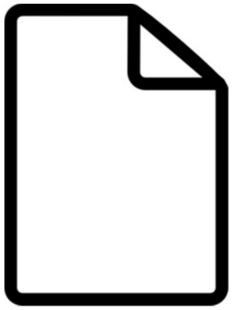
Prepared by Lo Jia Yuan (EA16062)

Supervised by MD Rizal bin Othman (Dr.)

INTRODUCTION

- Big data from sensor (mixture of image, sound, etc)
- Huge challenge to store and transmit
- Require high bandwidth, storage and time
- Data and information security
- Comparison to traditional compression algorithm?
- Lossless?
- Solution: Compression and reconstruction using DNN





PROBLEM STATEMENT

1. High bandwidth needed for transmitting big sensor data	2. Embedded devices have limited processing power and memory
3. Privacy concerns in data transmission	4. Time, cost, energy, accuracy trade offs

OBJECTIVES

1. To propose a structure of Deep Neural Network (DNN) for data compression and reconstruction
2. To study suitable training and model parameters for DNN data compressor
3. To evaluate the propose DNN using appropriate loss function and test methods

SCOPES

- Explore using TensorFlow framework and Python language
- Learn about different types of neural network
- Learn about different types of compression algorithm
- Learn about different methods of compression based on NN
- Build, design and train models that can compress and reconstruct inputs
- Use MNIST and other online database to train and evaluate the model

METHODOLOGY

- Method to build model:
  - Build model using Keras
  - Train model using MNIST [1] (28x28x1) and COCO [2] (128x128x3) datasets
  - Train model on computer
  - Test model on Raspberry Pi

METHODOLOGY

- Method to validate:
  - MSE - Mean Square Error
  - PSNR - Peak Signal to Noise Ratio (dB)
  - SSIM - Structural Similarity Index, L = dynamic range, k1 = 0.01, k2 = 0.03 [3]
  - MSSIM - Multiple Scale Structural Similarity Index [4]

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

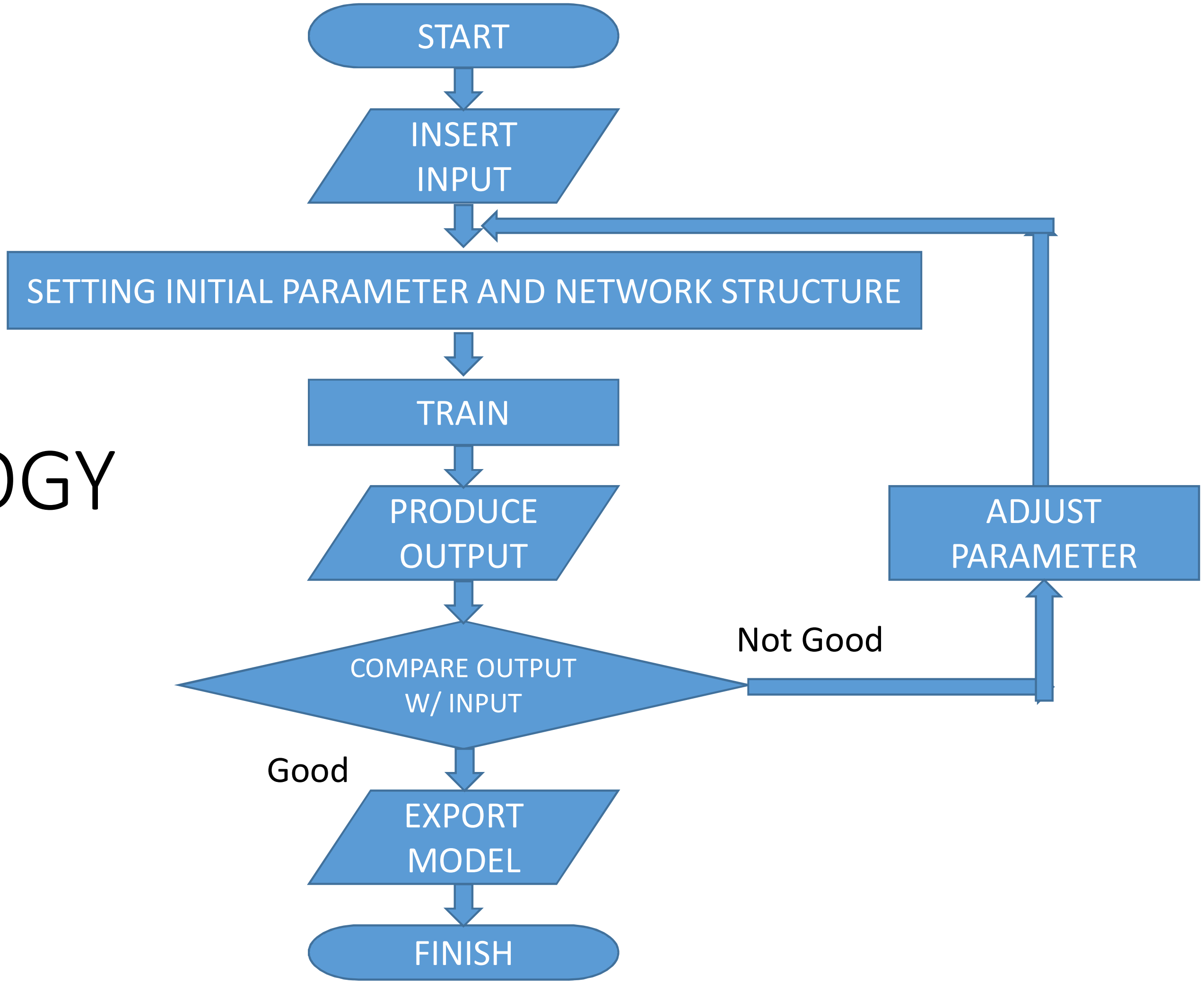
$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

$$c_1 = (k_1 L)^2 \quad c_2 = (k_2 L)^2$$

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) = 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)$$

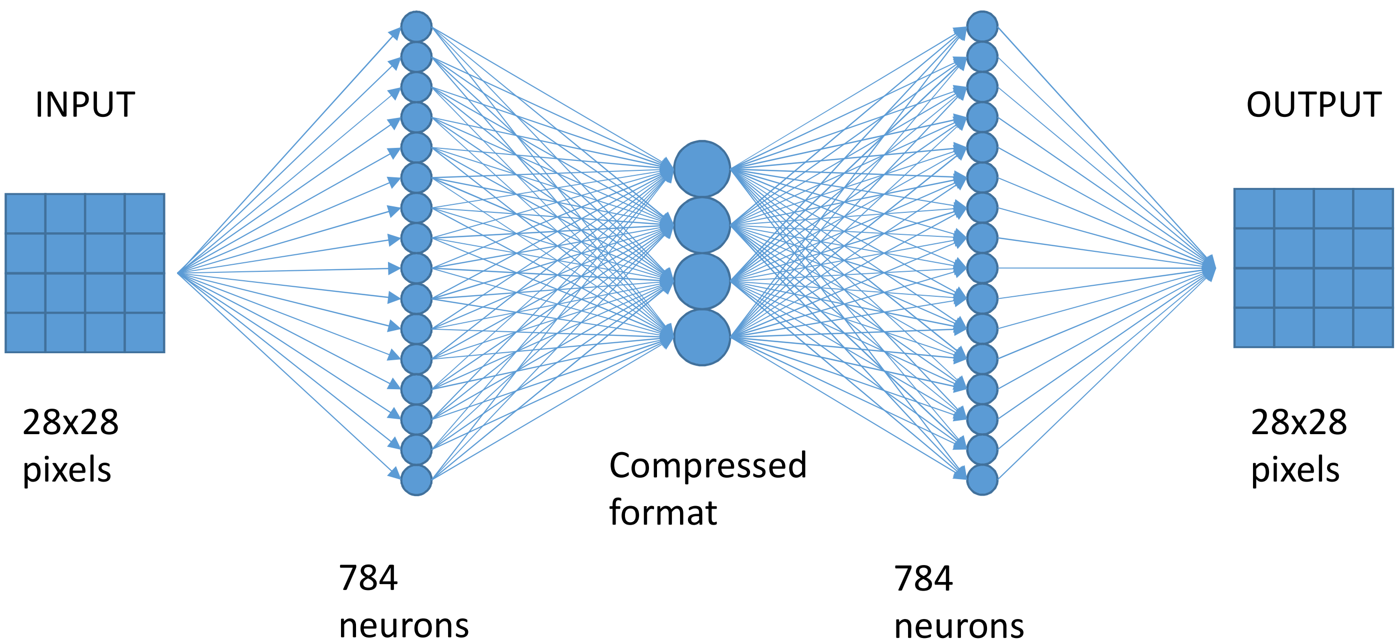
$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2 + \mu_y^2 + c_1} \quad c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{\sigma_x^2 + \sigma_y^2 + c_2} \quad s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3}$$

METHODOLOGY



METHODOLOGY

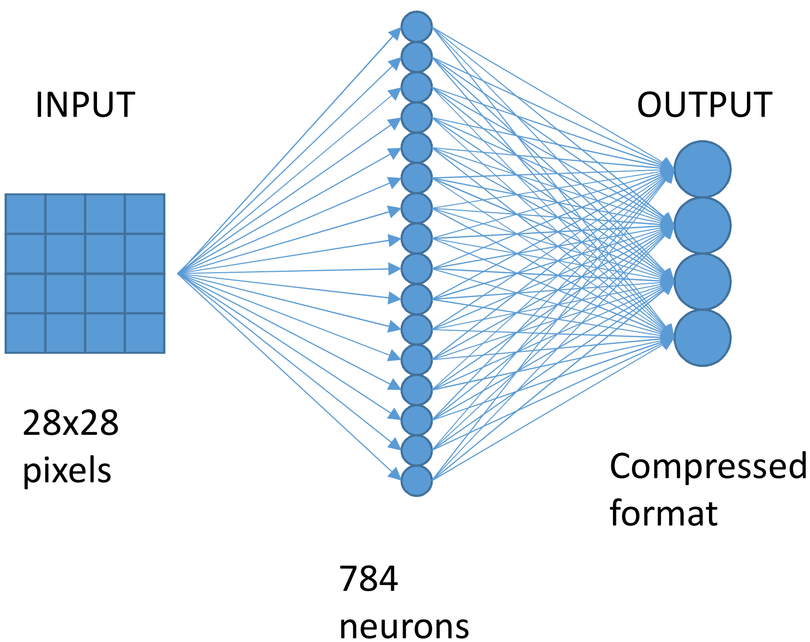
Example of model in training (autoencoder)



Training is done jointly for better result [5]

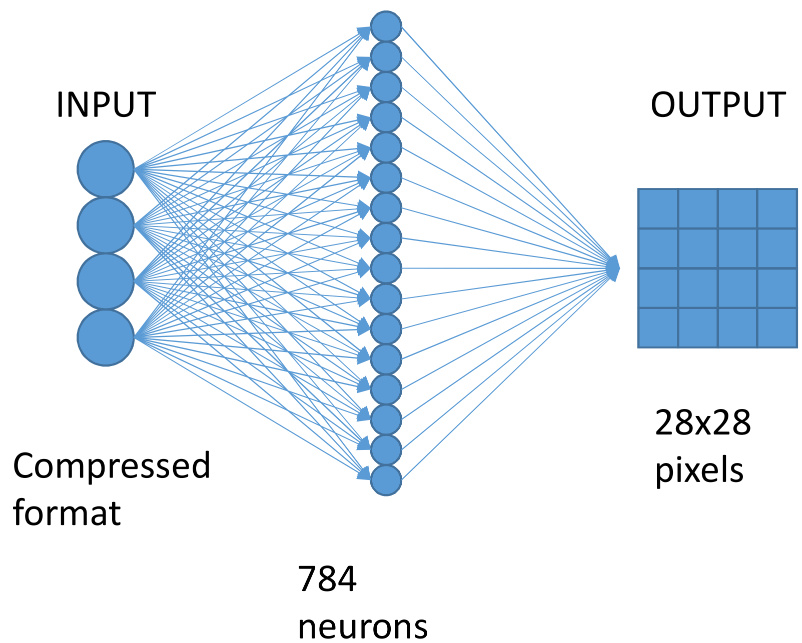
METHODOLOGY

Example of final model in use (encoder)



METHODOLOGY

Example of final model in use (decoder)

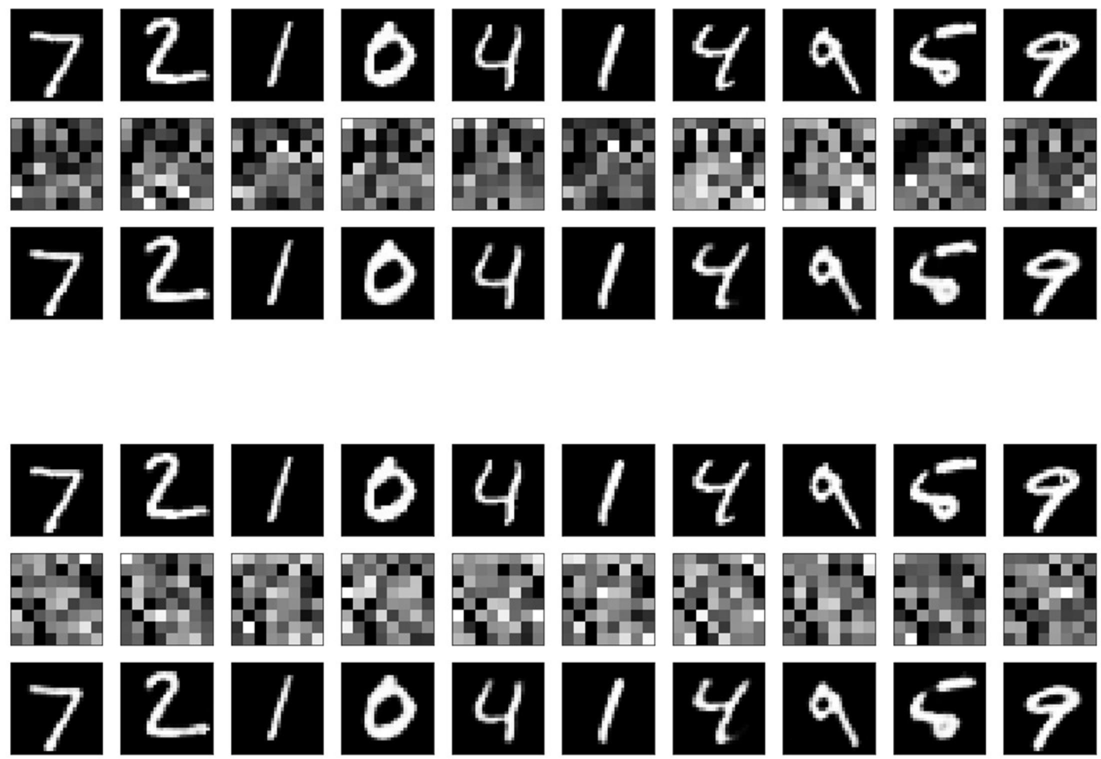


RESULT

RESULT (MNIST)

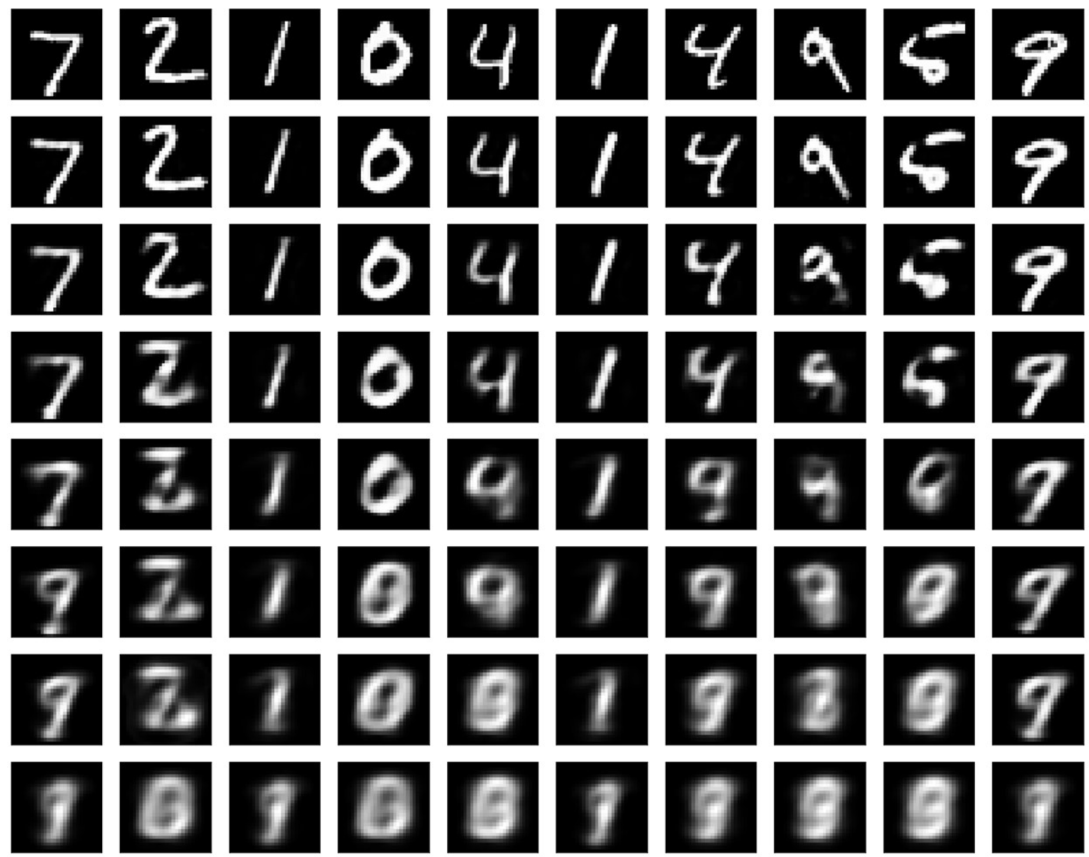
Each training will have different initial starting point and therefore different encoding and decoding results

So one encoded image cannot be used on another network to decode it



RESULT (MNIST) – One hidden layer

BOTTLENECK LAYER SIZE	
	Original (784)
	64
	32
	16
	8
	4
	2
	1



RESULT (MNIST) – MSE

	Original (784)										
BOTTLENECK LAYER SIZE	64	0.0018	0.0045	0.0010	0.0020	0.0035	0.0006	0.0050	0.0032	0.0049	0.0033
	32	0.0047	0.0129	0.0023	0.0090	0.0098	0.0012	0.0111	0.0193	0.0163	0.0081
	16	0.0114	0.0383	0.0052	0.0161	0.0167	0.0040	0.0239	0.0379	0.0352	0.0163
	8	0.0244	0.0586	0.0114	0.0214	0.0331	0.0110	0.0476	0.0471	0.0768	0.0338
	4	0.0339	0.0578	0.0160	0.0490	0.0386	0.0156	0.0516	0.0546	0.0813	0.0352
	2	0.0388	0.0603	0.0183	0.0537	0.0557	0.0180	0.0571	0.0615	0.0780	0.0411
	1	0.0508	0.0872	0.0311	0.0618	0.0562	0.0285	0.0581	0.0592	0.0792	0.0644

RESULT (MNIST) – PSNR (dB)

BOTTLENECK LAYER SIZE										
Original (784)										
64	27.44	23.46	30.11	27.01	24.51	32.38	22.97	24.95	23.06	24.77
32	23.29	18.91	26.31	20.45	20.08	29.07	19.53	17.15	17.88	20.90
16	19.45	14.17	22.84	17.94	17.78	23.99	16.22	14.21	14.54	17.88
8	16.12	12.32	19.42	16.71	14.80	19.60	13.23	13.27	11.15	14.71
4	14.70	12.38	17.95	13.10	14.14	18.08	12.88	12.63	10.90	14.54
2	14.11	12.20	17.37	12.70	12.54	17.45	12.44	12.11	11.08	13.86
1	12.94	10.59	15.07	12.09	12.50	15.45	12.36	12.28	11.01	11.91

RESULT (MNIST) – SSIM

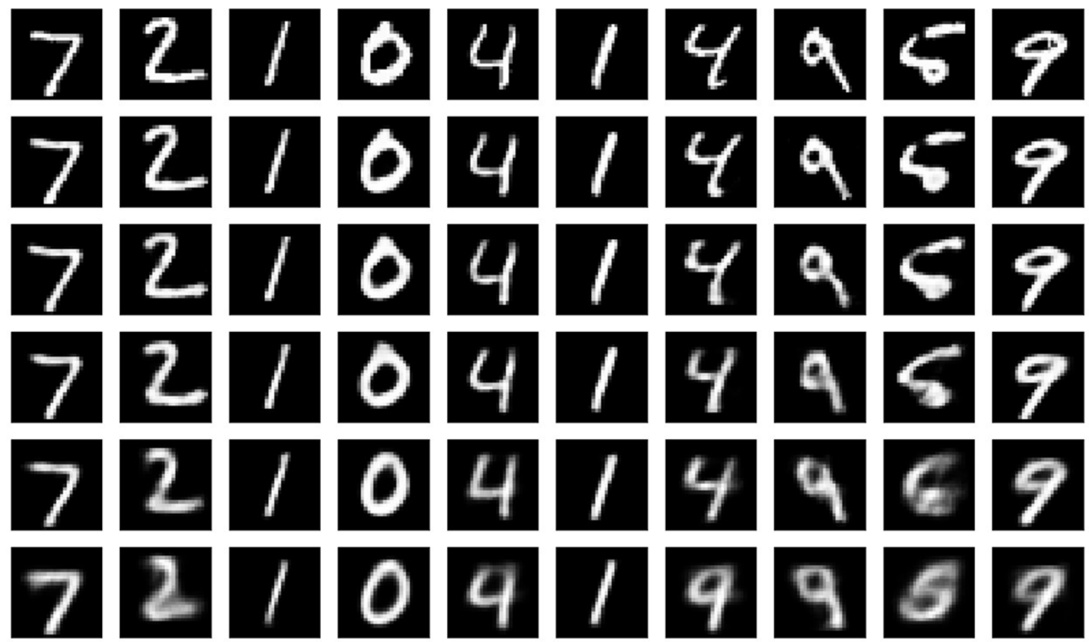
BOTTLENECK LAYER SIZE	Original (784)										
	64	0.9643	0.9590	0.9641	0.9813	0.9398	0.9793	0.9160	0.9430	0.9432	0.9698
	32	0.9147	0.8371	0.9038	0.9151	0.8352	0.9325	0.8361	0.6817	0.8094	0.9213
	16	0.7646	0.5488	0.7849	0.8468	0.6899	0.8354	0.6055	0.4570	0.6227	0.8093
	8	0.5447	0.3796	0.5834	0.7767	0.4149	0.6362	0.2455	0.3286	0.2087	0.6214
	4	0.4573	0.2892	0.4081	0.4795	0.2979	0.5250	0.2104	0.1610	0.1476	0.5952
	2	0.3857	0.2428	0.3230	0.4340	0.1804	0.4266	0.1498	0.0997	0.1599	0.5370
	1	0.1978	0.0502	0.1803	0.2976	0.1545	0.2693	0.1109	0.1325	0.1272	0.2645

RESULT (MNIST) – MSSIM

BOTTLENECK LAYER SIZE	Original (784)										
	64	0.9970	0.9942	0.9981	0.9972	0.9932	0.9990	0.9906	0.9943	0.9876	0.9913
	32	0.9862	0.9703	0.9950	0.9726	0.9636	0.9981	0.9744	0.9266	0.9579	0.9778
	16	0.9498	0.8628	0.9825	0.9292	0.9434	0.9900	0.8909	0.7736	0.8700	0.9498
	8	0.8544	0.6382	0.9411	0.9042	0.6124	0.9616	0.6467	0.5846	0.5175	0.7892
	4	0.7183	0.5042	0.9115	0.6390	0.5384	0.9322	0.5599	0.3219	0.1908	0.8264
	2	0.6480	0.4463	0.8745	0.6208	0.1592	0.9007	0.4673	0.2496	0.1482	0.8138
	1	0.3461	nan	0.7290	0.4390	0.1995	0.7951	0.4179	0.3019	0.1336	0.5610

RESULT (MNIST) – Multiple hidden layers

COMBINATION OF LAYERS	Original
	784-256-128-64-128…
	784-256-128-64-32-64…
	784-256-128-64-32-16-32…
	784-256-128-64-32-16-8-16…
	784-256-128-64-32-16-8-4-8…



RESULT (MNIST) - MSE

COMBINATION OF LAYERS	Original										
	784-256-128-64-128…	0.0024	0.0032	0.0007	0.0033	0.0032	0.0006	0.0050	0.0053	0.0052	0.0040
	784-256-128-64-32-64…	0.0034	0.0057	0.0009	0.0068	0.0049	0.0008	0.0091	0.0108	0.0132	0.0061
	784-256-128-64-32-16-32…	0.0042	0.0157	0.0014	0.0123	0.0079	0.0007	0.0199	0.0247	0.0266	0.0126
	784-256-128-64-32-16-8-16…	0.0066	0.0375	0.0018	0.0256	0.0159	0.0018	0.0297	0.0384	0.0567	0.0181
	784-256-128-64-32-16-8-4-8…	0.0171	0.0558	0.0053	0.0319	0.0210	0.0067	0.0591	0.0462	0.0892	0.0258

RESULT (MNIST) – PSNR (dB)

	Original										
COMBINATION OF LAYERS	784-256-128-64-128…	26.18	24.93	31.83	24.81	24.94	32.45	22.99	22.74	22.81	23.94
	784-256-128-64-32-64…	24.63	22.47	30.59	21.64	23.08	31.04	20.40	19.67	18.80	22.16
	784-256-128-64-32-16-32…	23.77	18.04	28.62	19.10	21.03	31.55	17.02	16.06	15.74	19.01
	784-256-128-64-32-16-8-16…	21.83	14.26	27.34	15.91	17.98	27.50	15.27	14.15	12.46	17.43
	784-256-128-64-32-16-8-4-8…	17.67	12.54	22.80	14.96	16.78	21.72	12.28	13.36	10.50	15.88

RESULT (MNIST) - SSIM

COMBINATION OF LAYERS	Original										
	784-256-128-64-128…	0.9772	0.9769	0.9898	0.9713	0.9576	0.9923	0.9354	0.9535	0.9424	0.9638
	784-256-128-64-32-64…	0.9624	0.9449	0.9864	0.9403	0.9312	0.9907	0.8899	0.8784	0.8349	0.9470
	784-256-128-64-32-16-32…	0.9361	0.8336	0.9779	0.8918	0.8953	0.9899	0.7283	0.6917	0.7225	0.8543
	784-256-128-64-32-16-8-16…	0.9055	0.5552	0.9734	0.7618	0.7242	0.9788	0.5394	0.5446	0.3731	0.7573
	784-256-128-64-32-16-8-4-8…	0.6877	0.3571	0.9114	0.7184	0.6458	0.9014	0.2335	0.4002	0.0990	0.6421

RESULT (MNIST) - MSSIM

COMBINATION OF LAYERS											
	Original										
	784-256-128-64-128…	0.9960	0.9969	0.9986	0.9953	0.9930	0.9988	0.9927	0.9928	0.9871	0.9901
	784-256-128-64-32-64…	0.9921	0.9832	0.9980	0.9851	0.9846	0.9982	0.9759	0.9707	0.9621	0.9828
	784-256-128-64-32-16-32…	0.9816	0.9431	0.9947	0.9555	0.9575	0.9967	0.9195	0.8979	0.8660	0.9565
	784-256-128-64-32-16-8-16…	0.9815	0.7511	0.9957	0.8855	0.8858	0.9944	0.8264	0.8070	0.6949	0.9315
	784-256-128-64-32-16-8-4-8…	0.9175	0.5774	0.9613	0.8687	0.8122	0.9764	0.6746	0.7032	nan	0.8636

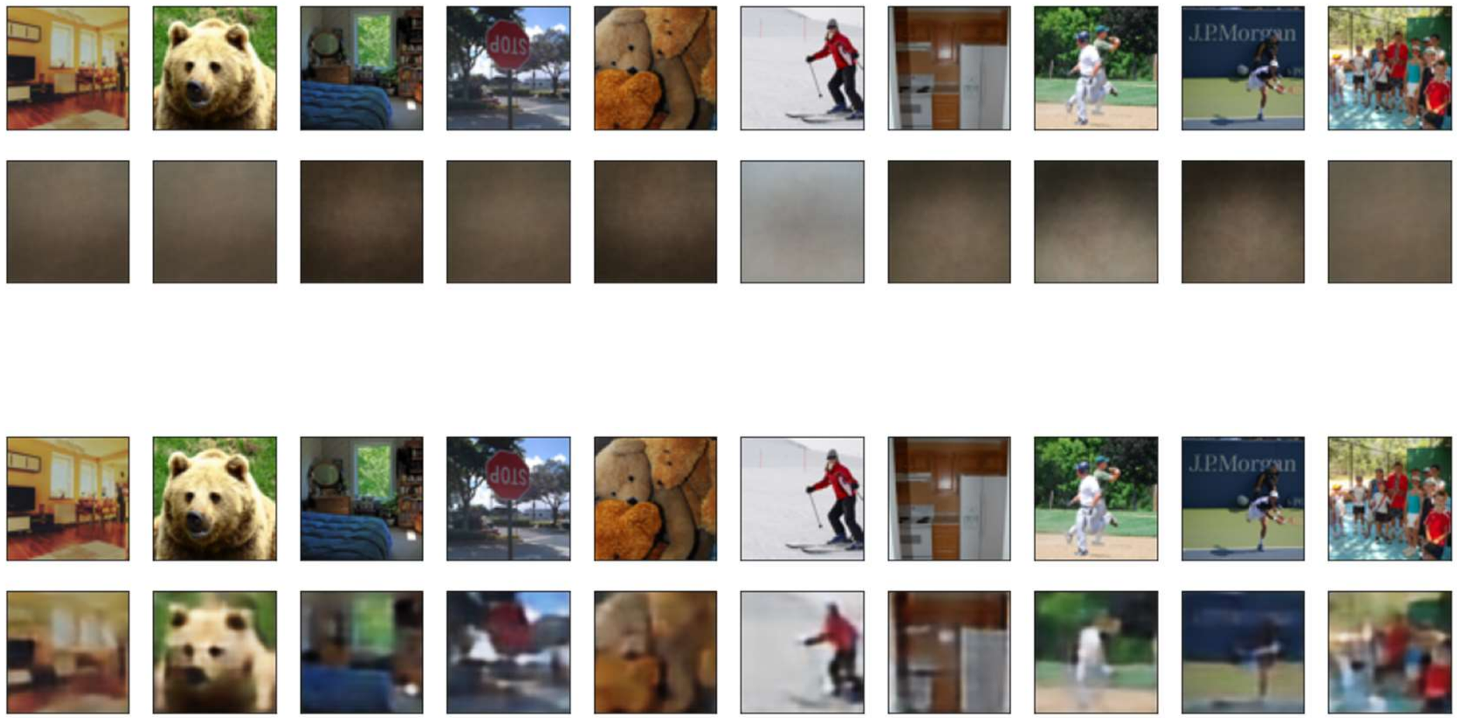
RESULT (COCO) – Dense vs Convolution

Fully connected Dense network

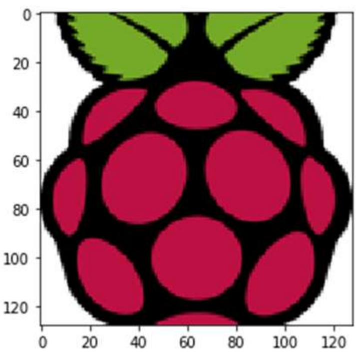
- Training time longer
- Bad result

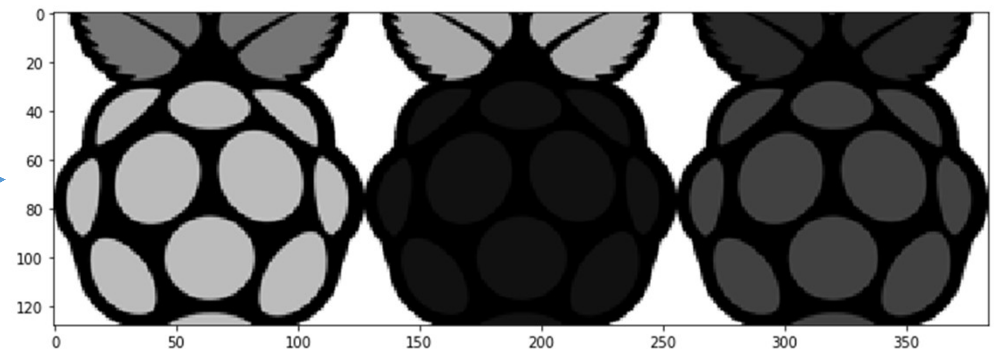
Convolution network

- Training time shorter
- Better result

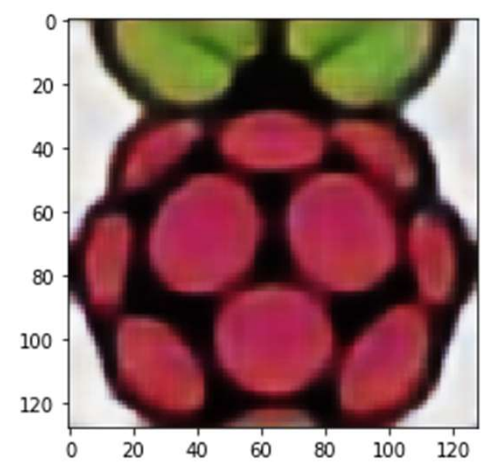


RESULT – Convolution





128x128x3
= 49152



16x16x16
= 4096

RESULT (COCO) – Dense vs Convolution

Model: "autoencoder"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 12288)]	0
dense (Dense)	(None, 9830)	120800870
dropout (Dropout)	(None, 9830)	0
dense_1 (Dense)	(None, 7864)	77310984
dense_2 (Dense)	(None, 6291)	49478715
dense_3 (Dense)	(None, 6291)	39582972
dense_4 (Dense)	(None, 7864)	49480288
dropout_1 (Dropout)	(None, 7864)	0
dense_5 (Dense)	(None, 9830)	77312950
dense_6 (Dense)	(None, 12288)	120803328

Total params: 534,770,107
Trainable params: 534,770,107
Non-trainable params: 0

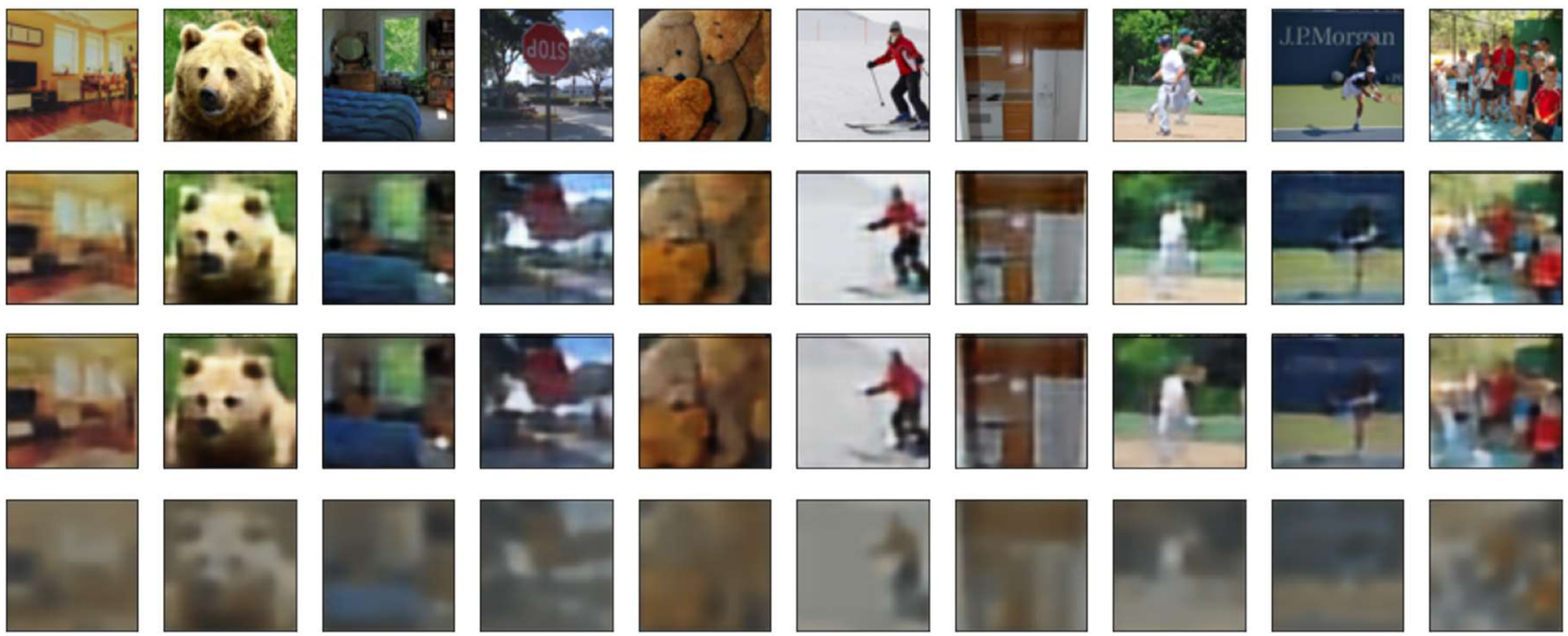
Model: "autoencoder"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 64, 64, 3)]	0
conv2d (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 16)	4624
max_pooling2d_1 (MaxPooling2	(None, 16, 16, 16)	0
conv2d_2 (Conv2D)	(None, 16, 16, 16)	2320
max_pooling2d_2 (MaxPooling2	(None, 8, 8, 16)	0
conv2d_3 (Conv2D)	(None, 8, 8, 16)	2320
up_sampling2d (UpSampling2D)	(None, 16, 16, 16)	0
conv2d_4 (Conv2D)	(None, 16, 16, 16)	2320
up_sampling2d_1 (UpSampling2	(None, 32, 32, 16)	0
conv2d_5 (Conv2D)	(None, 32, 32, 32)	4640
up_sampling2d_2 (UpSampling2	(None, 64, 64, 32)	0
conv2d_6 (Conv2D)	(None, 64, 64, 3)	867

Total params: 17,987
Trainable params: 17,987
Non-trainable params: 0

RESULT (COCO) - Dropout

Original
No dropout
Apply dropout down sample
Apply all layers with dropout



RESULT (COCO) – Dropout - MSE

Original										
No dropout	0.0044	0.0082	0.0045	0.0089	0.0034	0.0071	0.0031	0.0080	0.0047	0.0128
Apply dropout down sample	0.0063	0.0105	0.0073	0.0114	0.0049	0.0223	0.0051	0.0119	0.0070	0.0146
Apply all layers with dropout	0.0268	0.0477	0.0158	0.0281	0.0154	0.0871	0.0211	0.0421	0.0211	0.0367

RESULT (Neural Network)

```
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 128, 128, 64)      1792
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
dropout_5 (Dropout)          (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 64, 64, 32)        18464
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
dropout_6 (Dropout)          (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 32, 32, 16)        4624
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 16, 16, 16)        2320
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 32, 32, 16)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 32)        4640
_________________________________________________________________
up_sampling2d_5 (UpSampling2 (None, 64, 64, 32)        0
_________________________________________________________________
dropout_8 (Dropout)          (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
up_sampling2d_6 (UpSampling2 (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 128, 128, 3)       1731
=================================================================
Total params: 52,067
Trainable params: 52,067
Non-trainable params: 0
_________________________________________________________________
```

encoder

```
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         (None, 128, 128, 3)       0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 128, 128, 64)      1792
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 64, 64, 64)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 64, 64, 32)        18464
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 32, 32, 16)        4624
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 16, 16, 16)        0
=================================================================
Total params: 24,880
Trainable params: 24,880
Non-trainable params: 0
_________________________________________________________________
```

decoder

```
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         (None, 16, 16, 16)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 16, 16, 16)        2320
_________________________________________________________________
up_sampling2d_4 (UpSampling2 (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 32, 32, 32)        4640
_________________________________________________________________
up_sampling2d_5 (UpSampling2 (None, 64, 64, 32)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 64, 64, 64)        18496
_________________________________________________________________
up_sampling2d_6 (UpSampling2 (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 128, 128, 3)       1731
=================================================================
Total params: 27,187
Trainable params: 27,187
Non-trainable params: 0
_________________________________________________________________
```

DISCUSSION

- During FYP1, there were some mistakes in calculating PSNR
- Max value need to consider the pixel value is normalized (1.0) or not (255.0)

$$
\begin{aligned}
PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\
&= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right)
\end{aligned}
$$

DISCUSSION

- Neural network has fixed input sizes
- Datasets need to be preprocessed to ensure compatibility with the neural network
- Example: You cannot use images of size (28, 28, 1) on a network with input size (64, 64, 3)
- Challenges in ensuring the datasets compatible:
  - Time consuming
  - Require Python language skill

DISCUSSION

- Using fully connected Dense layers can be computationally expensive as the network scales
- For images, convolution layers can increase performance without computationally expensive
- While dropout may make reduce the performance the network [6]
    - In this case it can preserve the colour of the image

DISCUSSION (Future work & Challenges)

- Current implementation using Python and “import tensorflow” is slow
  - Convert to Tensorflow Lite can speed up the neural network performance on lower end hardware
- Neural network take a long time to train on CPU and requires expensive GPU from Nvidia
  - Wait for other GPU support
  - Use Google Colab / Pay for other service
- Current neural network may not be good enough
  - Rather than training to compress entire image, try training to obtain the features from the small pixel values
  - Variational autoencoder (VAE) may be another good alternative

CONCLUSION

- Neural network based data compression system is achievable
- Care needs to be taken to avoid loss of information
- More work need to be done to improve performance
- Dropout can improve autoencoder performance
- IoT can use neural network to speed up encoding and decoding with negligence loss
- One neural network trained is not the same with another, thus privacy concern can be dismissed

REFERENCES

[1]Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[2]T. Lin et al., "Microsoft COCO: Common Objects in Context", *arXiv.org*, 2020. [Online]. Available: https://arxiv.org/abs/1405.0312. [Accessed: 22- Jul- 2020].

[3]Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004, doi: 10.1109/TIP.2003.819861.

[4]Z. Wang, E. P. Simoncelli and A. C. Bovik, "Multiscale structural similarity for image quality assessment," The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003, Pacific Grove, CA, USA, 2003, pp. 1398-1402 Vol.2, doi: 10.1109/ACSSC.2003.1292216.

[5]Y. Zhou, D. Arpit, I. Nwogu and V. Govindaraju, "Is Joint Training Better for Deep Auto-Encoders?", *arXiv.org*, 2020. [Online]. Available: https://arxiv.org/abs/1405.1380. [Accessed: 22- Jul- 2020].

[6]N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Jmlr.org*, 2020. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html. [Accessed: 22- Jul- 2020].