

Department of Informatics

INDIVIDUAL ASSIGNMENT

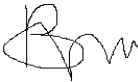
Surname	Bhunu							
Initials	M							
Student Number	2	3	9	5	3	0	0	5
Module Code	INF				7	9	1	
Assignment number	2							
Name of Lecturer	Dr. Mike Nkongolo							
Date of Submission	2024/10/11							
<p>Declaration:</p> <p>I declare that this assignment, submitted by me, is my own work and that I have referenced all the sources that I have used.</p>								
Signature of Student								

TABLE OF CONTENTS

1.	INTRODUCTION	1
2.	DATA COLLECTION	1
3.	PREPROCESSING THE DATA	2
3.1	NORMALIZATION.....	2
3.2	RESHAPING THE DATA.....	2
3.3	DATA SPLITTING	2
4.	EXPLORATORY DATA ANALYSIS (EDA)	2
4.1	DATA VISUALIZATIONS	2
4.2	PIXEL INTENSITY DISTRIBUTION	3
4.3	ASSESSING THE IMAGE BRIGHTNESS.....	3
4.4	RELATIONSHIPS BETWEEN PIXEL VALUES ACROSS THE RGB COLOUR CHANNELS.....	3
5.	LSTM AUTOENCODER MODEL	4
5.1	ENCODER.....	4
5.2	LATENT SPACE	4
5.3	DECODER.....	4
5.4	MODEL COMPILATION AND TRAINING	4
5.4.1	Training and validation loss.....	4
5.5	EVALUATION ON TEST DATA	5
5.6	ANALYSIS OF RESULTS	5
6.	CODE EXPLANATION	6
6.1	DATA PROCESSING CODE	6
6.2	LSTM AUTOENCODER CODE	6
6.3	MODEL TRAINING CODE.....	7
7.	CONCLUSION.....	8
8.	REFERENCES.....	10

ANALYSIS FOCUS

Application of the Long Short-Term Memory model to process CIFAR dataset. Can the LSTM learn from the images while preserving the essential features?

LIST OF FIGURES

Figure 1 Autoencoders in action (Yang, 2024)	1
Figure 2 Original Images before reconstruction attempt.	2
Figure 3 Pixel density after normalization.....	3
Figure 4 Correlation matrix showing correlations in the color channels.	3
Figure 5 Scatter plot matrix showing correlations.	4
Figure 6 Training and validation loss.	5
Figure 7 Results from an attempt to reconstruct the images.	5

1. INTRODUCTION

An autoencoder is a type of neural network (NN) that learns mainly from compressed data, encoded representation of the data for the purposes of size reduction or feature extraction (Yang, 2024). The introduction of autoencoders with their ability to learn unsupervised representations have been on the forefront of demystifying image processing. The medical industry for instance, has reaped the rewards of autoencoders through the removal of noise from MRI images. Autoencoders can deal with tasks such as image denoising, compression, reconstruction, anomaly detection and generative image modelling. When it comes to image processing, autoencoders work by compressing the images, keeping the necessary features, and then reproducing the images as they were before (Yang, 2024). Essentially, they work as a mirror of the images.

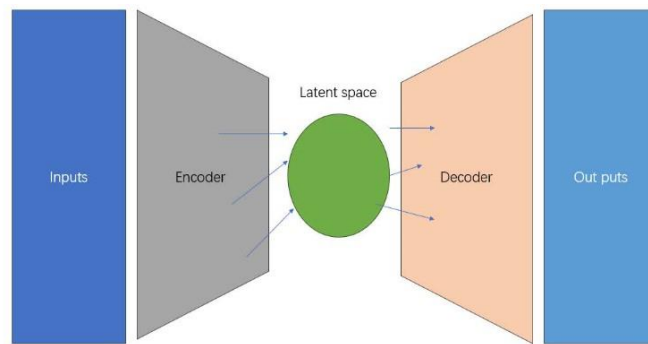


Figure 1 Autoencoders in action (Yang, 2024)

In this technical report, we will be focusing on the Long Short-Term Memory (LSTM) autoencoder to reconstruct images from the CIFAR-10 dataset that have been distorted. LSTM is a type of Recurrent Neural Network (RNN) that is capable of effectively processing and predicting sequential data overcoming most of the issues the RNN has like the vanishing gradient problem (Arifin *et al.*, 2023).

The CIFAR-10 is one of the most known datasets in ML, it consists of 60,000 32 by 32 images that are classified into 10 categories. Most ML engineers turn to this dataset because of its size and diversity, and it has proven to be a good way to benchmark tasks in ML.

The main objective of this technical report is to reconstruct clear images from the blurred versions from the CIFAR-10 dataset. To achieve this, we will train a LSTM autoencoder to learn all the important features from the images and use this representation to recover the original quality of images.

2. DATA COLLECTION

To complete the implementation of the LSTM autoencoder, the CIFAR-10 dataset was used to implement the neural network. The CIFAR-10 dataset as mentioned before consists of 60,000 images 32 by 32 classified into 10 different classes. The images in the dataset are all colour images with 6,000 images per class. 50,000 images from

the dataset are all training images and 10,000 images are for testing. To simply import the data, the following script was used.

```
(x_train, _), (x_test, _) = cifar10.load_data()
```

The dataset is imported through tensorflow.keras.datasets which is a library that has ready-made datasets that are ready to use.

3. PREPROCESSING THE DATA

3.1 NORMALIZATION

When preparing data for neural networks, normalization is an essential step as it ensures that the input is scaled in such a way that the model can handle efficiently (Xu *et al.*, 2024). The pixel values for the CIFAR-10 images that were in the range of [0,255], were normalized by dividing them by 255. The new scale of [0,1] is efficient for the training of neural networks.

3.2 RESHAPING THE DATA

The LSTM models are designed to process sequential data in the form of sequences (Viswambaran *et al.*, 2024). Thus, the dataset must be reshaped to meet the requirements of the LSTM autoencoder. The images were reshaped into a sequence of 32 rows each containing 32 sets of 3 values that correspond to the red, green, and blue (RGB) channels.

3.3 DATA SPLITTING

For the purposes of evaluating the model, the dataset was split into two the training dataset and testing dataset. An 80% training and 20% for validation. Splitting in ML ensures that the model is validated on the data that it did not see during the training to check on a models level of generalization (Jain *et al.*, 2022).

4. EXPLORATORY DATA ANALYSIS (EDA)

4.1 DATA VISUALIZATIONS

To gain better understanding of the dataset, we visualized sample images from the dataset. The images represent 10 different classes, that are a diverse range of objects such as airplanes, cars, and animals.

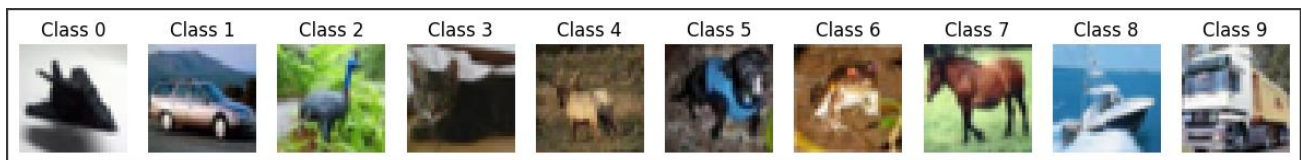


Figure 2 Original Images before reconstruction attempt.

4.2 PIXEL INTENSITY DISTRIBUTION

After normalization, the following histogram was plotted to check the distribution of the pixel intensities in the dataset. The pixel values are scaled between 0 and 1. The representations helps to identify if the values were distributed between the [0,1] range.

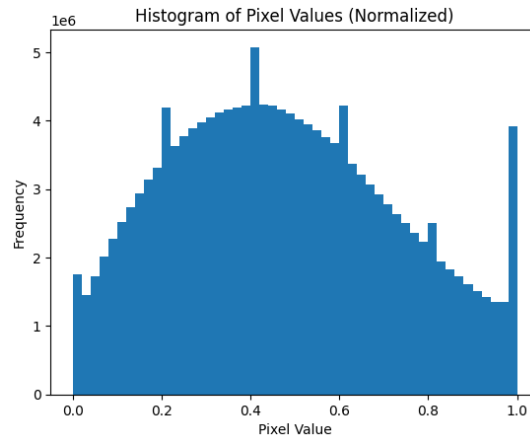


Figure 3 Pixel density after normalization.

4.3 ASSESSING THE IMAGE BRIGHTNESS

The mean variance was calculated to access the overall brightness of the images. The mean value of the pixel intensities after normalization was approximately 0.47 and the variance was 0.063.

4.4 RELATIONSHIPS BETWEEN PIXEL VALUES ACROSS THE RGB COLOUR CHANNELS

A corelation matrix was generated to examine the relationships between pixel values across the RGB colour channels. The matrix just shows the relationship between colour intensities in the dataset. The high levels of correlations between the color channels indicate that the images have a higher rate of predictability across the color channel. The LSTM in this case is able to predict pixels.

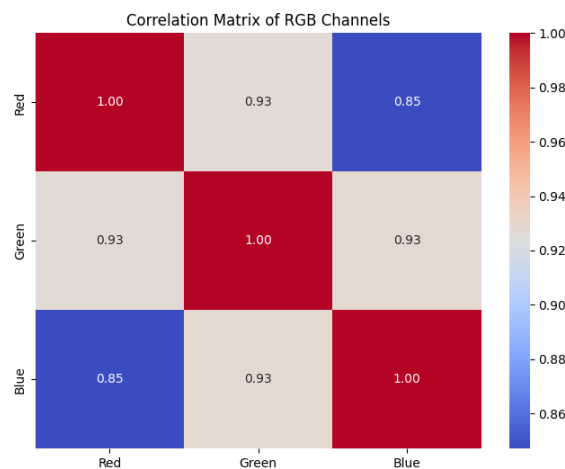


Figure 4 Correlation matrix showing correlations in the color channels.

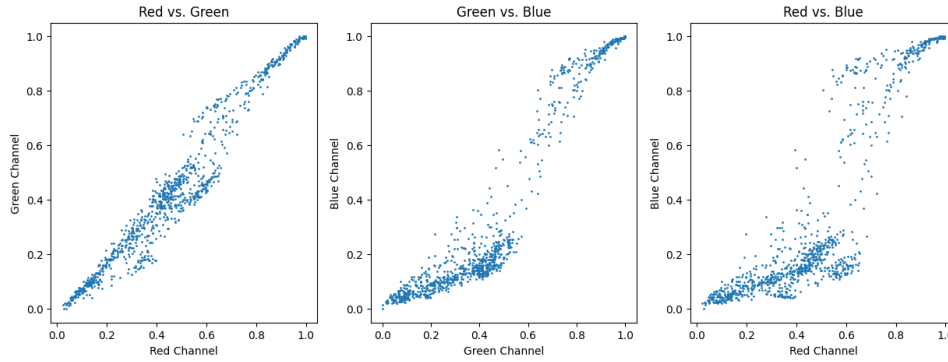


Figure 5 Scatter plot matrix showing correlations.

5. LSTM AUTOENCODER MODEL

5.1 ENCODER

According to Kvalheim and Sontag, 2024, an encoder is defined as a function that maps input data from higher-dimensional space to a lower dimensional space. In other words, the encoder is the part of the autoencoder that is responsible for compressing the images and extracting the important information from the images while compressing the images into a compact latent space representation. The LSTM encoder in this case uses the tanh activation to learn more about the image and consists of two layers.

5.2 LATENT SPACE

The latent space represents the compressed representation of the input image that contains the most relevant learn features by the encoder, it saves as a bridge between the input data and the reconstructed data (Chen and Guo, 2023). This representation allows the model to reconstruct the image while keeping the most important information removing the irrelevant information.

5.3 DECODER

The decoder is responsible for reconstructing the original image from the above-mentioned latent space (Kvalheim and Sontag, 2024).

5.4 MODEL COMPILATION AND TRAINING

The Mean Squared Error function is the characterization between the actual and the predicted values (Blot, 2020). The model was compiled using the Mean Squared Error function that basically calculates the average squared difference between the original and reconstructed images. For the purposes of training the Adam optimizer was used for training.

```
autoencoder.compile(optimizer=Adam(learning_rate=0.0001), loss='mse')
```

5.4.1 TRAINING AND VALIDATION LOSS

The training data was validated over 200 epochs. The training and loss validation loss decreases steadily, this indicated that the model was learning to reconstruct the images effectively. The validation loss also stabilized

after about 175 epochs, meaning that the model had reached its optimal performance.

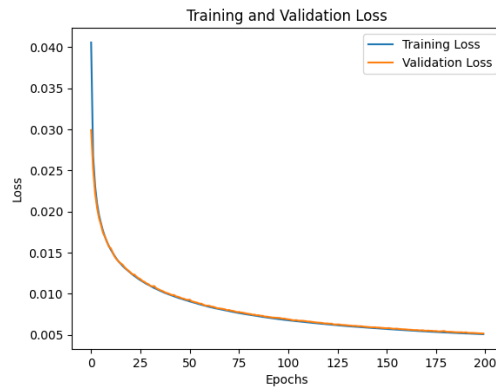


Figure 6 Training and validation loss.

5.5 EVALUATION ON TEST DATA

After training the model, its performance was evaluated on the test set, the final test loss was computed using the Mean Squared Error (MSE), metric, it measures just how well the model reconstructed the images compared to the original one. The test loss on the model was 0.0051. The snippet below was used to evaluate the evaluation of the test data.

```
test_loss = autoencoder.evaluate(x_test, x_test)
print(f"Test Loss: {test_loss}")
```

5.6 ANALYSIS OF RESULTS

To evaluate the model, the original images were compared to the constructed images. The images show how well the LSTM autoencoder was very effective in constructing the images.

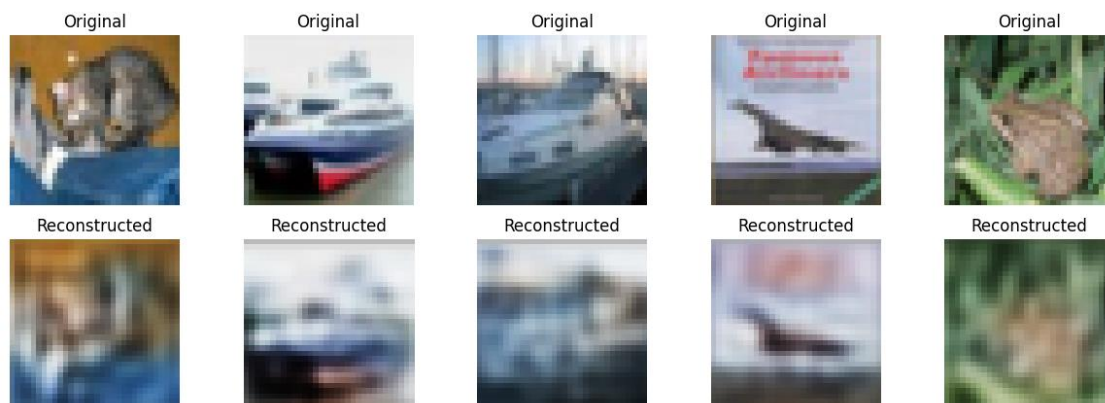


Figure 7 Results from an attempt to reconstruct the images.

The training and validation loss graph showed compelling results that suggested that the model had little to no loss as the training went on. Unfortunately, the images were constructed but did not give compelling results based

on how well the actual model performed. Tuning to the LSMT was done such as providing more layers of the encoder and decoders but the images got better after every adjustment but still would not give the accurately extracted images. The following section does show some of the code snippets that were implemented with explanation for the encoder and decoder. The additional layers suggest that these images should have been produced without any blurry effects.

6. CODE EXPLANATION

6.1 DATA PROCESSING CODE

This involves the process of loading the CIFAR-10 dataset using Keras library through a built-in function, normalizing the data by scaling the pixel range [0,1] through dividing each value by 255 and reshaping the images for LSTM input as a sequence of rows (32) with 96 features. The following snippet represents the steps taken.

Load CIFAR-10 dataset

```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

Normalize the data: scale pixel values to the range [0, 1]

```
x_train = x_train.astype('float32') / 255.0
```

```
x_test = x_test.astype('float32') / 255.0
```

Reshape the data for LSTM input (sequence length 32, features 32*3 for RGB channels)

Each image is reshaped into a sequence of 32 rows and 96 features (32*3 for RGB)

```
x_train = x_train.reshape(-1, 32, 32 * 3)
```

```
x_test = x_test.reshape(-1, 32, 32 * 3)
```

6.2 LSTM AUTOENCODER CODE

The autoencoder consists of two main parts, the encoder, and the decoder. The encoder compresses the image inputs into a smaller latent representation using an LSTM. The decoder goes ahead and takes the latent representation and reconstructs the image row by row using another LSTM. The following code snippets and explanations represent the functionality of the autoencoder. However, this LSTM uses two layers for both the encoder and decoder. The purpose was to come up with a smaller representation of the latent space.

Input structure.

LSTM autoencoder model with increased latent space size and added layers.

```
input_img = Input(shape=(32, 32 * 3))
```

The input takes in 32 rows of 96 features in this case. Each of the rows represent the 32 by 32 image from the snippet above.

Encoder with two LSTM layers

```
encoded = LSTM(512, activation='tanh', return_sequences=True)(input_img)
encoded = LSTM(256, activation='tanh')(encoded)
```

The encoder uses two layers for the purposes of further compression of the data. The first layer has 512 units which process the input sequentially. The second layer further compresses the data input with 256 units. The compression from the two layers makes it slightly easier for the model to understand the data that is in the latent representation. The activation, tanh in this case is used to learn the compact representation of the model.

Latent space representation

```
latent_space = RepeatVector(32)(encoded)
```

After compression, the data is repeated 32 times using the Repeat Vector. This step makes sure that the data is ready for the decoder during reconstruction.

Decoder with two LSTM layers

```
decoded = LSTM(256, activation='tanh', return_sequences=True)(latent_space)
decoded = LSTM(32 * 3, activation='sigmoid', return_sequences=True)(decoded)
```

The Decoder follows the same steps as those of the encoder, the first LSTM processes the spaces that was repeated by the repeat vector. The second layer outputs 32 by 3 (96 features) and the sigmoid activation function values stay between the 0 and 1 range.

6.3 MODEL TRAINING CODE

The model was trained with a batch size of 128 for 200 epochs. The validation set consists of 20% of the data and it used to monitor the model's performance.

Training the model for 200 epochs, using early stopping to stop when validation loss stops improving.

```
x_train, x_val, y_train, y_val = train_test_split(x_train, x_train, test_size=0.2, random_state=42)
history = autoencoder.fit(x_train, x_train, epochs=200, batch_size=128, validation_data=(x_val, x_val),
callbacks=[early_stopping])
```

After training, the model's performance is evaluated on the test set using the MSE, which is the difference between the original and the constructed images.

```
test_loss = autoencoder.evaluate(x_test, x_test)
```

```
print(f"Test Loss: {test_loss}")
```

Finally, the following code snippet compared the original images against the plotted images for the purposes of showing the difference between the original and the constructed images. This also helps to evaluate our LSTM autoencoder. The snippet will only show 5 of the images.

Visualizing the original and reconstructed images

```
n = 5
plt.figure(figsize=(15, 5))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(32, 32, 3))
    plt.title("Original")
    plt.axis("off")
    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(reconstructed[i].reshape(32, 32, 3))
    plt.title("Reconstructed")
    plt.axis("off")
plt.show()
```

7. CONCLUSION

The Goal of the project was to reconstruct images from the CIFAR10 dataset using the LSTM autoencoder. The model managed to achieve a test loss of 0.0051. The model showed effectiveness in constructing images. The model's competency was also represented by the images that were reproduced after constructing the images.

One of the major advantages of the LSTM Autoencoder is its ability to process sequential data, this made it suitable for dealing with images as sequences of pixel rows. Unfortunately, the produced images were distorted and blurry. The model was trained over and over again and it still failed to generate images that were easy to see on the human eye.

The use of Convolution autoencoders maybe suitable for this type of application as opposed to the LSTM. improvements can be made to the LSTM by tuning the hyperparameters in the architecture. To further

experiment with the LSTM, it may be useful to perform tasks such as image denoising or anomaly detection in the images.

The LSTM autoencoder shows, overall potential in the application of the field of image processing, by tuning the parameters, it may prove to be a very useful tool when dealing with image processing.

8. REFERENCES

- Arifin, S. et al. (2023) 'Long Short-Term Memory (LSTM): Trends and Future Research Potential', *International Journal of Emerging Technology and Advanced Engineering*, 13, pp. 24–35. Available at: https://doi.org/10.46338/ijetae0523_04.
- Blot, M. (2020) 'A Characterization of Mean Squared Error for Estimator with Bagging', *ArXiv [Preprint]*. Available at: https://www.academia.edu/91351947/A_Characterization_of_Mean_Squared_Error_for_Estimator_with_Bagging (Accessed: 11 October 2024).
- Chen, S. and Guo, W. (2023) 'Auto-Encoders in Deep Learning—A Review with New Perspectives', *Mathematics*, 11(8), p. 1777. Available at: <https://doi.org/10.3390/math11081777>.
- Jain, E. et al. (2022) 'A Diagnostic Approach to Assess the Quality of Data Splitting in Machine Learning'. arXiv. Available at: <https://doi.org/10.48550/arXiv.2206.11721>.
- Kvalheim, M.D. and Sontag, E.D. (2024) 'Why should autoencoders work?' arXiv. Available at: <https://doi.org/10.48550/arXiv.2310.02250>.
- Viswambaran, R.A. et al. (2024) 'Automatic Design of LSTM Networks with Skip Connections through Evolutionary and Differentiable Architecture Search', in *2024 IEEE Congress on Evolutionary Computation (CEC). 2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. Available at: <https://doi.org/10.1109/CEC60901.2024.10611997>.
- Xu, S. et al. (2024) 'On the Preprocessing of Physics-informed Neural Networks: How to Better Utilize Data in Fluid Mechanics'. arXiv. Available at: <http://arxiv.org/abs/2403.19923> (Accessed: 11 October 2024).
- Yang, W. (2024) 'Autoencoder in Machine Learning', *Science and Technology of Engineering, Chemistry and Environmental Protection*, 1(8). Available at: <https://doi.org/10.61173/p7hxp58>.