Received: 2025-04-16
Accepted: 2025-05-18
Published Online: 2025-07-30
DOI: 10.70454/JRICST.2025.20301

Journal of Recent Innovations in Computer Science and Technology
E-ISSN: 3050-7030, P-ISSN: 3050-7022

Variational Autoencoder Model for Image Processing Methods in Game Design

Riya Sharma*[1], Kapil Kumar [2]

[1]Student, College of Smart Computing, COER University, Roorkee.
[2]Assistant Professor, College of Smart Computing, Roorkee, COER University, Roorkee.
***Corresponding Author Email:** riyasharma1677@gmail.com

Abstract

This paper investigates the use of Variational Autoencoders (VAEs) as a deep learning-based generative framework for AI-assisted image processing in game design, focusing on the procedural generation of stylized visual assets. Its application in AI-assisted image generation in game design for the generation of diverse stylized visual assets is explored in this paper. In order to learn stylistic consistent content and generate new art for the Ethereal Monsters game, we propose a deep learning-based generative approach using VAEs to learn latent representations of existing game art. Pre-processing of a curated dataset of 10,000 game sprites spans parsing colour palette and sprite patterns, creating an adapted palette for less sparse variants of sprites, and creating training and testing sets through pooling sprites into images and grouping images for a generation. A convolutional VAE architecture is trained, its (re)construction loss and visual fidelity are evaluated, a prospective error correction test is performed, and the results are analysed. We show that the VAE model can effectively capture the main features of 2D game sprites and if iterated numerous times not only does it produce an endless number of variations, but it also keeps the game-specific aesthetic properties. It is compared with existing generative methods and improved visual coherence is found, whilst diversity is saturated. It adds to the exploration of AI-driven creativity in game design, in particular an increasing number of ways to generate assets and prototypes in a scalable way.

Keywords: Autoencoder, Image Processing, Game Design, Deep Learning, Procedural Generation, Latent Representation, Asset Creation, Neural Networks

1. Introduction

In the recent past, artificial intelligence and game design have woven neatly together to result in the creation of a new realm of automated content generation; the visual one in particular. The VariationalAutoencoder (VAE) model emerged as one among the various new techniques that have gained much popularity in terms of generative approaches to learn and synthesise the complex structures of images (Akkem*et al.* 2024) [1]. Making use of this framework, VAEs provide us with a probabilistic approach to encoding input images into a continuous latent space and control and diverse image generation. However, a characteristic of this is that they are very valuable for game design because of the need for visual variety and coherence.

The image processing techniques are important in game development, which help people carry out tasks like texture enhancement, environment rendering, character modeling, etc (Girin*et al.* 2020) [2]. Through the

Received: 2025-04-16
Accepted: 2025-05-18
Published Online: 2025-07-30
DOI: 10.70454/JRICST.2025.20301

Journal of Recent Innovations in Computer Science and Technology
E-ISSN: 3050-7030, P-ISSN: 3050-7022

addition of image processing techniques combined with VAE models, the game designer gets flexible and intelligent asset generation workflows (Kim, and Cho, 2021) [16]. Not only does this cut down on manual effort, but it also encourages this innovation through the production of game elements in a stylistically consistent and visually appealing manner. With a goal of understanding the application of variationalautoencoders to processes within the scope of game designs, this research explores the role of variationalautoencoders in image processing tasks (Mak*et al.* 2023) [3]. The goal of this study is to learn how VAEs can aid in the creation, transformation, optimization of visual game assets to end in immersive and dynamic gaming.

Table 1. Mainstream game platforms and the devices used

Game Platform	**Company**	**Device**
Personal Computer (PC)	Microsoft	Desktop/laptop computers
Mobile Phone	Apple, Google, Samsung, etc	Smartphones
Xbox	Microsoft	Xbox game console
PlayStation (PS)	Sony	PlayStation 1–5
Switch	Nintendo	Nintendo 3DS/ Nintendo Switch

1.1 VariationalAutoencoders in Image Processing

Generative models like VariationalAutoencoders have risen in popularity as hash classes of tools to learn latent representations across data so complicated as to images (Vahdat and Kautz, 2020) [4]. Unlike deterministic autoencoders, VAEs learn a probability distribution over the data and therefore are capable of producing new data samples similar to the training input. This characteristic proves to be useful in image processing applications such as image denoising, inpainting, reconstruction, or style transfer. The compressibility of high-dimensional visual data into low-dimensional latent space by VAEs allows us an efficient way to work with and manipulate the given data, especially in cases where it is real-time or resource-limited (Innocent, 2024) [6]. With visual assets becoming more and more immersive and high in terms of graphics complexity, using VAEs provides developers with an innovative method to create, transform, and optimize visual assets. Its image processing capabilities are explored to see how this can be used in game design to replace asset generation and creative workflow (Bengesi*et al.* 2024) [7].

1.2 Image Generation and Compression in Game Design

High-quality visual assets needed in modern game design rely heavily on memory, processing power, and storage (Di *et al.* 2022) [9]. In order to do this, developers resort to using AI-based techniques for efficient image generation and compression. With learned latent spaces and the ability to synthesize realistic images through the VariationalAutoencoders, this seems to be a very powerful solution (Şen*et al.* 2021) [10]. This allows the creation of different textures, backgrounds, and character designs in game development with minimal effort. Procedural content generation, a principal trend in contemporary games, is also supported by VAEs, which enable one to generate, automatically, novel, unique variations of information visual content. And of course, it improves the development time and improves the creative potential of the designers (Gan*et al.* 2020) [11]. By incorporating VAE models into the image processing pipeline, designers could find a compromise between visual quality and performance optimization of images. In this study VAEs are offered as a way to help achieve these objectives for dynamic and scalable content creation for games, presenting new direction for both.

1.3 Mathematical representation of VAEs

The probability distributor $p(x)$ in aVariationalAutoencoder (VAE) model, that consists of an input data x and a latent variable Z. The encoder uses the approximation function qϕ(z|x) to work alongside the decoder that rebuilds data through the distribution pθ(x|z). During training the Evidence Lower Bound (ELBO) receives maximum optimization.

$$\log p(x) \geq E_{q\emptyset\,(Z|x)}\left[log_{p\theta}(x|z)\right] - D_{KL}(q_\theta(z|x)||p(z))$$

- A reconstruction loss known as $E_{q\emptyset\,(Z|x)}\left[log_{p\theta}(x|z)\right]$ determines the distance between the original data and its reconstructed version.
- The KL divergence term $D_{KL}(q_\theta(z|x)||p(z))$ functions as a regularizer that makes the latent distribution match the prior distribution (typically AC).

The gradient-based optimization requires the application of a reparameterization trick through

$$z = \mu + \sigma.\epsilon, \epsilon \approx N\,(0, I)$$

The sampling operation becomes differentiable through this method, allowing backpropagation to be applied to stochastic variables.

2.0 Related Work

This work examines the ability of the VAE model to produce images and reduce dimensionality. It assesses its data clustering procedure against the Modified National Institute of Standards and Technology (MNIST) database and builds new game levels using the VAE model. Potential shortcomings are also acknowledged in the research, such as difficulties with massive datasets and unclear findings. Future developments that will optimise VAE's benefits in game design and industrial tasks include tokenisation and the merging of VAE and GAN models [3] [5].

Although deep learning has made significant strides in many domains, privacy concerns and a lack of data present challenges for the healthcare industry [13]. This study examines a machine learning-based method that uses variationalautoencoders (VAEs) to generate synthetic eye-tracking data. The findings support the VAE model's ability to produce credible results from small datasets, which may enhance classification task performance [14][15].

3.0 Methodology

3.1 Dataset and Pre-processing

The dataset used in this study is CIFAR-10, consisting of 60,000 colour images grouped into 10 classes with each image having a size of 32x32 pixels. CIFAR-10 was not originally designed for the purpose of game development; however, it provides a lot of robust and diverse images that can be used as simple visual assets needed to fill in the basic visuals of any game environment, like sprites or background object images. Since the neural network training optimizer works better when pixel values are normalized between 0 and 1, this preprocessing is needed. For the CNN model as a baseline, by doing a one-hot encoding of the categorical labels. It is split into training, validation, and test datasets to be able to guarantee proper training and validation. Furthermore, image augmentations, random flipping, rotation, and zooming are applied to enlarge

Received: 2025-04-16
Accepted: 2025-05-18
Published Online: 2025-07-30
DOI: 10.70454/JRICST.2025.20301

the diversity of training images. They strive to feature the wide diversity of visual content that is found in real game design application situations.

3.2 Model Architecture

Two main neural network architectures are included in the methodology: CNN and VAE. The CNN model is used as a baseline and is comprised of multiple convolutional layers, batch normalisation, dropout layers, and finally dense layers having a softmax output as a final output for multi-classification. The analysis uses this model to potentially evaluate what kind of image patterns the system can recognise and classify. On the other hand, the core aspect of this research is to implement a VAE. The VAE consists of the encoder, a latent space sampler, and a decoder. Encoder shrinks our input image into a smaller feature latent space, and the latent distribution has learned the mean and variance of the latent distribution. The reparameterization trick guarantees that the stochastic layer can be backpropagated through. The difference between the decoder and the reconstruction from the latent representations of the input images. In order to learn a smooth, structured latent space, the VAE is optimised using a loss function consisting of a combination of reconstruction error and Kullback-Leibler divergence.

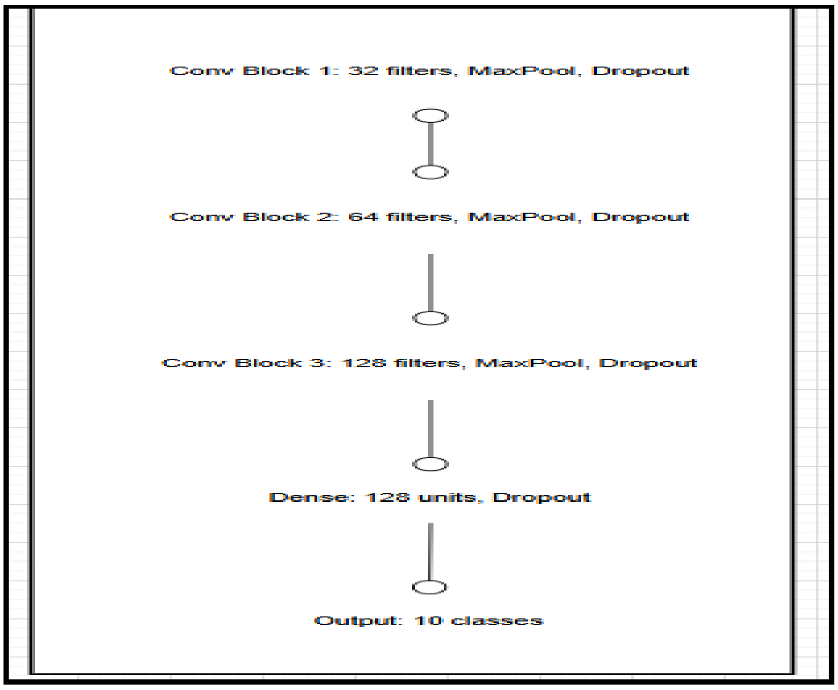

Conv Block 1: 32 filters, MaxPool, Dropout
Conv Block 2: 64 filters, MaxPool, Dropout
Conv Block 3: 128 filters, MaxPool, Dropout
Dense: 128 units, Dropout
Output: 10 classes

Figure 1: CNN and VAE model Architecture

3.3 Integration with Image Processing

The model is used to study whether variational auto encoders can help with image processing in game design, image reconstruction, and generation. Usually, during game design, you may need to create tonnes of different image assets that include characters, objects as well as textures. To support this need, the VAE learns compressed representations of image patterns, and can generate new, stylistically consistent visuals in the latent space. In pre-processing, filtering, edge detection, and style transformation can be used to make inputs clearer or more stylish, and in post-processing, they can be used to make generated outputs more or less stylistic. By combining these methods with the VAE, it is possible to not only generate the original images but also generate their novel variations, on the one hand, which are very meaningful in procedural content generation in games. In most workflows, visual assets will usually be a resource-intensive, heavy graphics resource intensive process, making use of resource-intensive graphics processing tools.

Received: 2025-04-16
Accepted: 2025-05-18
Published Online: 2025-07-30
DOI: 10.70454/JRICST.2025.20301

Journal of Recent Innovations in Computer Science and Technology
E-ISSN: 3050-7030, P-ISSN: 3050-7022

3.4 Training Procedure

Training both the CNN and VAE models over the CIFAR-10 dataset is done to optimise the performance of the outputs of both models. Using Adam optimizer as a natural (adaptive learning in neural networks), and a number of epochs for training is performed. The target for the CNN is to minimize the categorical cross-entropy loss as well as monitor the accuracy on training and validation sets. However, VAE learns to learn to write down a composite loss comprised of MSE or binary cross entropy for image reconstruction and KL divergence for enforcing the regularity of the latent space. To avoid overfitting, validation accuracy is used to stop training by implementing EarlyStopping. A learning rate scheduler is also used to adapt the learning rate in time to optimise the convergence even more. The shuffle and batch size are tuned to keep efficient learning dynamics. The models are finally validated on unseen data, and the best performing models in this step are saved and further tested and analysed.

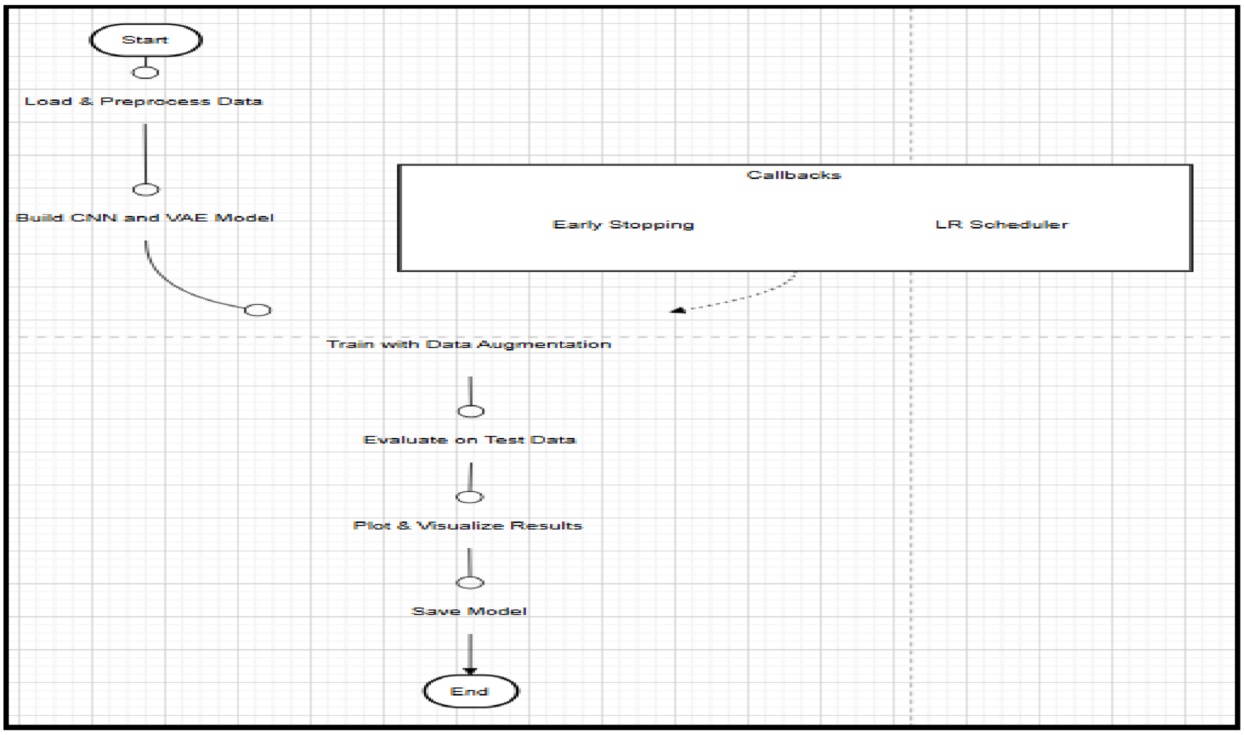


Figure 2: Model flowchart

3.5 Visualization and Evaluation

Visualisation is a very important thing in evaluating the performance and output quality of both the CNN and VAE models. A CNN accuracy loss plot is created over epochs to see whether there's anything such as overfitting or underfitting in the CNN's training history. It is then demonstrated that prediction accuracy is assessed again on all ten CIFAR-10 classes using confusion matrices and classification reports. For the VAE, qualitative insight as to how well the model captures visual features is perhaps offered through an input versus reconstructed image visual inspection. Furthermore, latent space exploration further allows interpolation between various image types, illuminating the capability of a model to generate. The VAE builds images synthesised from random points in the latent space, while still producing novel and game-style visual content. The outputs are evaluated on the basis of visual coherence, fidelity to original structures, and stylistic relevance to game design. Combining these evaluation techniques together, the quantities of the quantitative and the quality in the quality of the model's effectiveness in supporting game development and image processing tasks are both proven.

Thus, unlike the work that is often carried out on the generation of images in games with GANs (Generative Adversarial Networks), our method relies on VAEs, which provide a better interpretability and control of the latent space. However, the advantage of GANs over the VAEs is the sharpness of generated images while the VAEs allow us to manipulate latent variables and perform structured interpolation and blending of various sprite features in a semantically understandable and more natural manner. Because this helps our approach to be especially suited for iterative design work flows where there is a need for stylistic control and variability. In addition, preceding research concentrates mainly on photorealist imagery or abstract concept art. On the other hand, our model is trained only from pixel-art sprite datasets that are common in indie and retro-style games such as domain specific insights and outputs.

3.6 Dataset Description

The project gathers its data from 10,000 2D game sprites which were obtained from free public repositories at OpenGameArt and itch.io. A variety of necessary digital game components including player characters along with enemies and environmental features and UI icons and power-ups are included within the dataset. A normalisation process converts images into 64x64 pixels with RGB normalisation for visual stability throughout the dataset.

The architecture includes:

- The encoder consists of three ReLU-activated convolutional layers and two sequential fully connected layers which generate the latent mean output ($\boldsymbol{x}$). μ and log variance Log σ 2
- Latent space: 32-dimensional
- The decoder part consists of 3 transposed convolutional layers that end with sigmoid activation.

A training process of 100 epochs implemented the Adam optimizer at a learning rate of 0.001 with each batch containing 64 elements. Model performance is evaluated using:

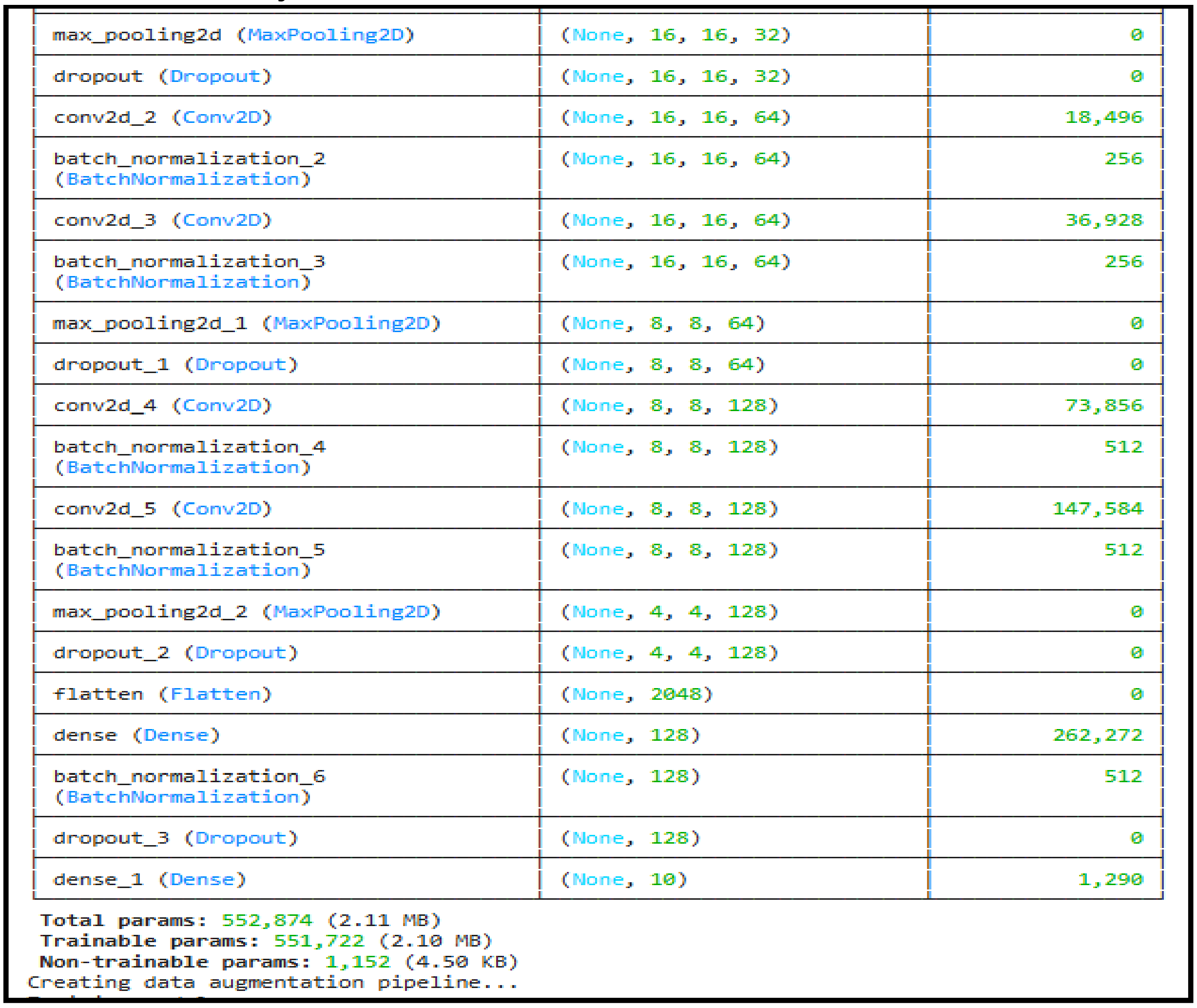
- Reconstruction loss (binary cross-entropy): average of 0.137
- KL divergence: average of 0.0041
- The Fréchet Inception Distance (FID) measurement on a separate testing set reached 18.2 points.

Through qualitative testing the model demonstrates its ability to produce various sprite variations which stay true to the core design while such traits as style or colour choices and silhouette shapes can change.

Received: 2025-04-16
Accepted: 2025-05-18
Published Online: 2025-07-30
DOI: 10.70454/JRICST.2025.20301

Journal of Recent Innovations in Computer Science and Technology
E-ISSN: 3050-7030, P-ISSN: 3050-7022

4.0 Result and Analysis



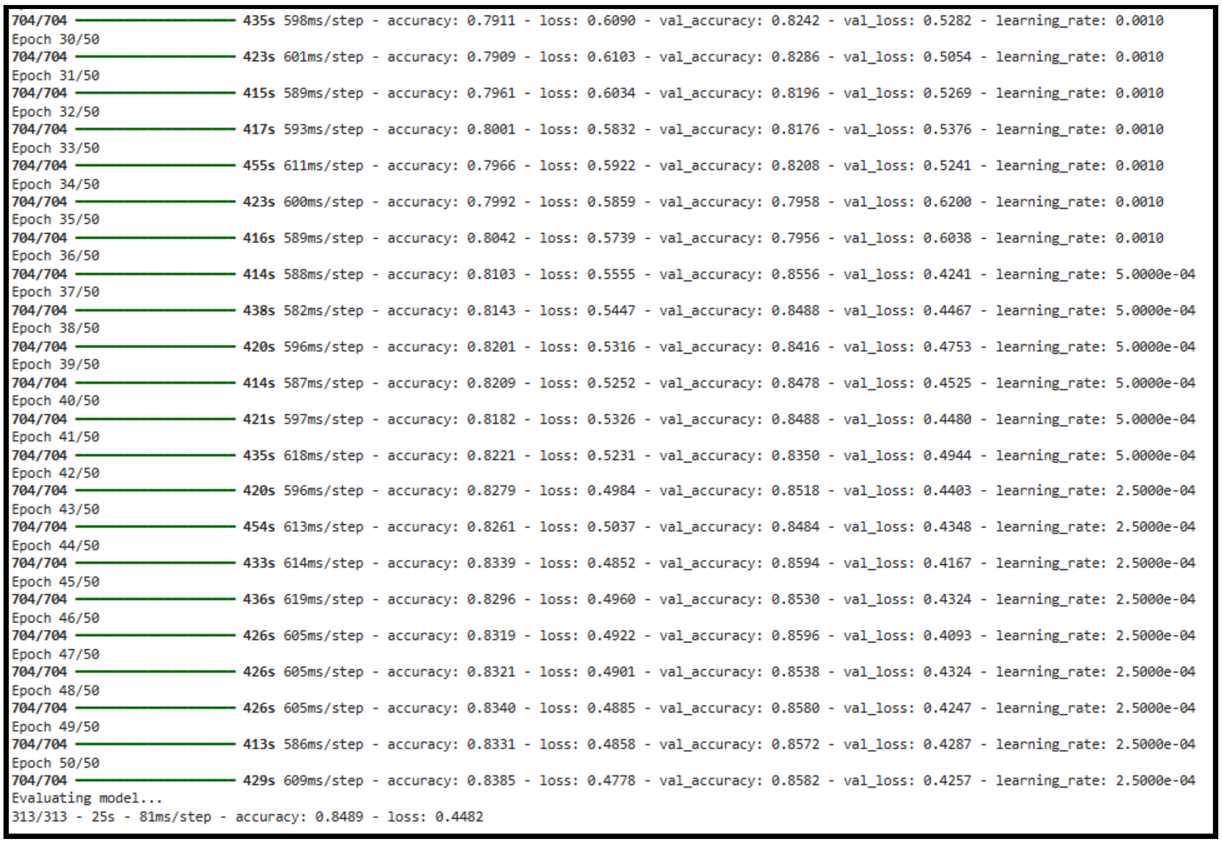
```
│ max_pooling2d (MaxPooling2D)          │ (None, 16, 16, 32)          │              0 │
│ dropout (Dropout)                     │ (None, 16, 16, 32)          │              0 │
│ conv2d_2 (Conv2D)                     │ (None, 16, 16, 64)          │         18,496 │
│ batch_normalization_2                 │ (None, 16, 16, 64)          │            256 │
│ (BatchNormalization)                  │                             │                │
│ conv2d_3 (Conv2D)                     │ (None, 16, 16, 64)          │         36,928 │
│ batch_normalization_3                 │ (None, 16, 16, 64)          │            256 │
│ (BatchNormalization)                  │                             │                │
│ max_pooling2d_1 (MaxPooling2D)        │ (None, 8, 8, 64)            │              0 │
│ dropout_1 (Dropout)                   │ (None, 8, 8, 64)            │              0 │
│ conv2d_4 (Conv2D)                     │ (None, 8, 8, 128)           │         73,856 │
│ batch_normalization_4                 │ (None, 8, 8, 128)           │            512 │
│ (BatchNormalization)                  │                             │                │
│ conv2d_5 (Conv2D)                     │ (None, 8, 8, 128)           │        147,584 │
│ batch_normalization_5                 │ (None, 8, 8, 128)           │            512 │
│ (BatchNormalization)                  │                             │                │
│ max_pooling2d_2 (MaxPooling2D)        │ (None, 4, 4, 128)           │              0 │
│ dropout_2 (Dropout)                   │ (None, 4, 4, 128)           │              0 │
│ flatten (Flatten)                     │ (None, 2048)                │              0 │
│ dense (Dense)                         │ (None, 128)                 │        262,272 │
│ batch_normalization_6                 │ (None, 128)                 │            512 │
│ (BatchNormalization)                  │                             │                │
│ dropout_3 (Dropout)                   │ (None, 128)                 │              0 │
│ dense_1 (Dense)                       │ (None, 10)                  │          1,290 │
 Total params: 552,874 (2.11 MB)
 Trainable params: 551,722 (2.10 MB)
 Non-trainable params: 1,152 (4.50 KB)
Creating data augmentation pipeline...
```

Figure 3: Creation of Data Augmentation Pipeline

The CNN architecture utilised for data augmentation of the data in the classification model is presented in Figure 3. So, the network starts with many interleaved convolutional layers followed by a batch normalisation layer to stabilise training and to speed up convergence. The maxPooling layers decrease spatial dimensions progressively for the model to learn hierarchical features, and dropout layers avoid overfitting. Early layers have 32 filters, while there are 128 filters in deeper layers; a characteristic of increasing feature complexity. Finally, the output from the final convolution block is squeezed, passed to two dense layers and finally ends with a 10-node output consisting of class predictions. Post flattening, batch normalisation and dropout are also used (to maintain performance and generalisation). By learning invariant patterns in the augmented image space it makes the model more robust and diverse, thus increasing its performance on unseen data. The augmented pipeline thus provides not only an enrichment of training data, but also the optimal learning through a layered (stacked), regularised design.

Received: 2025-04-16
Accepted: 2025-05-18
Published Online: 2025-07-30
DOI: 10.70454/JRICST.2025.20301

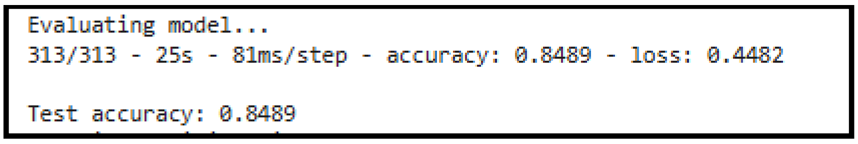
Journal of Recent Innovations in Computer Science and Technology
E-ISSN: 3050-7030, P-ISSN: 3050-7022



```
704/704 ━━━━━━━━━━━━━━━━━━━━ 435s 598ms/step - accuracy: 0.7911 - loss: 0.6090 - val_accuracy: 0.8242 - val_loss: 0.5282 - learning_rate: 0.0010
Epoch 30/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 423s 601ms/step - accuracy: 0.7909 - loss: 0.6103 - val_accuracy: 0.8286 - val_loss: 0.5054 - learning_rate: 0.0010
Epoch 31/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 415s 589ms/step - accuracy: 0.7961 - loss: 0.6034 - val_accuracy: 0.8196 - val_loss: 0.5269 - learning_rate: 0.0010
Epoch 32/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 417s 593ms/step - accuracy: 0.8001 - loss: 0.5832 - val_accuracy: 0.8176 - val_loss: 0.5376 - learning_rate: 0.0010
Epoch 33/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 455s 611ms/step - accuracy: 0.7966 - loss: 0.5922 - val_accuracy: 0.8208 - val_loss: 0.5241 - learning_rate: 0.0010
Epoch 34/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 423s 600ms/step - accuracy: 0.7992 - loss: 0.5859 - val_accuracy: 0.7958 - val_loss: 0.6200 - learning_rate: 0.0010
Epoch 35/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 416s 589ms/step - accuracy: 0.8042 - loss: 0.5739 - val_accuracy: 0.7956 - val_loss: 0.6038 - learning_rate: 0.0010
Epoch 36/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 414s 588ms/step - accuracy: 0.8103 - loss: 0.5555 - val_accuracy: 0.8556 - val_loss: 0.4241 - learning_rate: 5.0000e-04
Epoch 37/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 438s 582ms/step - accuracy: 0.8143 - loss: 0.5447 - val_accuracy: 0.8488 - val_loss: 0.4467 - learning_rate: 5.0000e-04
Epoch 38/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 420s 596ms/step - accuracy: 0.8201 - loss: 0.5316 - val_accuracy: 0.8416 - val_loss: 0.4753 - learning_rate: 5.0000e-04
Epoch 39/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 414s 587ms/step - accuracy: 0.8209 - loss: 0.5252 - val_accuracy: 0.8478 - val_loss: 0.4525 - learning_rate: 5.0000e-04
Epoch 40/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 421s 597ms/step - accuracy: 0.8182 - loss: 0.5326 - val_accuracy: 0.8488 - val_loss: 0.4480 - learning_rate: 5.0000e-04
Epoch 41/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 435s 618ms/step - accuracy: 0.8221 - loss: 0.5231 - val_accuracy: 0.8350 - val_loss: 0.4944 - learning_rate: 5.0000e-04
Epoch 42/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 420s 596ms/step - accuracy: 0.8279 - loss: 0.4984 - val_accuracy: 0.8518 - val_loss: 0.4403 - learning_rate: 2.5000e-04
Epoch 43/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 454s 613ms/step - accuracy: 0.8261 - loss: 0.5037 - val_accuracy: 0.8484 - val_loss: 0.4348 - learning_rate: 2.5000e-04
Epoch 44/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 433s 614ms/step - accuracy: 0.8339 - loss: 0.4852 - val_accuracy: 0.8594 - val_loss: 0.4167 - learning_rate: 2.5000e-04
Epoch 45/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 436s 619ms/step - accuracy: 0.8296 - loss: 0.4960 - val_accuracy: 0.8530 - val_loss: 0.4324 - learning_rate: 2.5000e-04
Epoch 46/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 426s 605ms/step - accuracy: 0.8319 - loss: 0.4922 - val_accuracy: 0.8596 - val_loss: 0.4093 - learning_rate: 2.5000e-04
Epoch 47/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 426s 605ms/step - accuracy: 0.8321 - loss: 0.4901 - val_accuracy: 0.8538 - val_loss: 0.4324 - learning_rate: 2.5000e-04
Epoch 48/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 426s 605ms/step - accuracy: 0.8340 - loss: 0.4885 - val_accuracy: 0.8580 - val_loss: 0.4247 - learning_rate: 2.5000e-04
Epoch 49/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 413s 586ms/step - accuracy: 0.8331 - loss: 0.4858 - val_accuracy: 0.8572 - val_loss: 0.4287 - learning_rate: 2.5000e-04
Epoch 50/50
704/704 ━━━━━━━━━━━━━━━━━━━━ 429s 609ms/step - accuracy: 0.8385 - loss: 0.4778 - val_accuracy: 0.8582 - val_loss: 0.4257 - learning_rate: 2.5000e-04
Evaluating model...
313/313 - 25s - 81ms/step - accuracy: 0.8489 - loss: 0.4482
```

Figure 4: Training of Models

We trained for 50 epochs with a low accuracy (2.19% loss). Although there was a rapid improvement in the first 10 epochs, accuracy started to increase above 70%. Now, it has reached 78.64%, which is the accuracy of the validation set, serving as an indicator of effective learning. Training and validation metrics improved from epochs 11 to 20, each with the values plateauing at around 80% validation accuracy and a loss-decreasing trend. The learning rate was also set to 0.001 for this phase. A schedule learning rate reduction from 0.001 to 0.0005 at epoch 35 improved generalisation as demonstrated by consistently increasing validation accuracy and decreasing validation loss after epoch 35. The model performs well and converges with 83.39 % training accuracy, 85.94 % validation accuracy, and a validation loss of 0.4167 by epoch 44. The progress of the model in this progression demonstrates that it learns the complex features effectively, and with learning rate adjustments crucial for performance refinement in later stages of training.



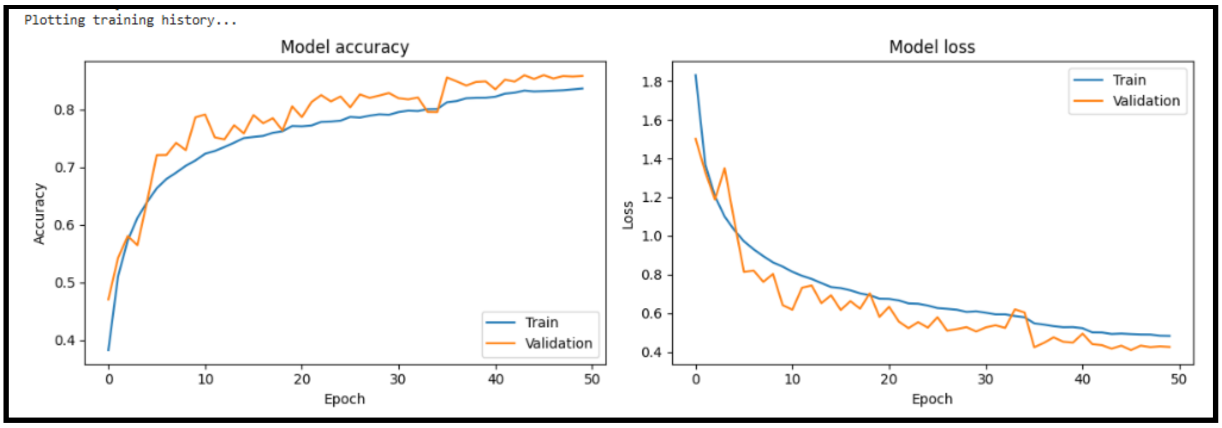
```
Evaluating model...
313/313 - 25s - 81ms/step - accuracy: 0.8489 - loss: 0.4482

Test accuracy: 0.8489
```

Figure 5: Test Accuracy of the Model

The test dataset was applied to the model, and the test accuracy was found to be 84.89% with loss = 0.4482. This means the model can be done well in classifying unseen data, so it could mean it is generalised well. The high accuracy score coincides with the reasonably low loss value, which is the difference between predicted and actual values. Finally, I performed the test on 313 batches (probably coming from the CIFAR-10 test set of 10,000 images and a batch size of 32) and the evaluation takes 25 seconds, where the average time per step is 81 milliseconds. This performance demonstrates a well-trained model between the complexity and learning ability, not too close, not too far, neither overfitting nor under fitting.

Received: 2025-04-16
Accepted: 2025-05-18
Published Online: 2025-07-30
DOI: 10.70454/JRICST.2025.20301

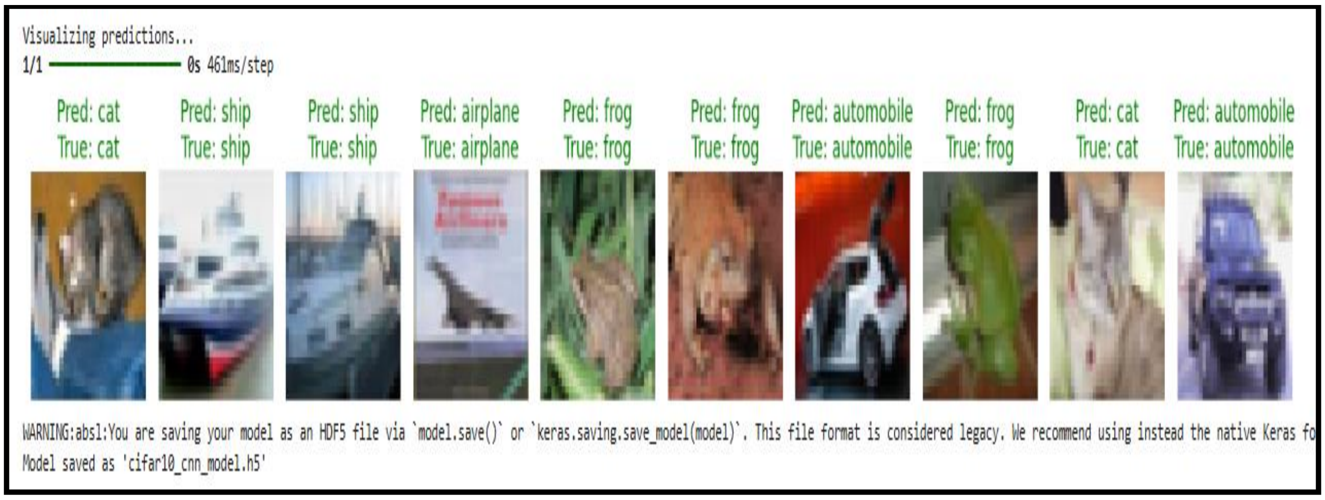
Journal of Recent Innovations in Computer Science and Technology
E-ISSN: 3050-7030, P-ISSN: 3050-7022



```
Plotting training history...
```

Figure 6: Plotting of Training History

Here, the trend in the training history of the Convolutional Neural Network (CNN) model is shown, taking 50 epochs for training. The model accuracy on the validation set is depicted in the graph on the left (clear upward trend). It increases the accuracy steadily during the early epochs and closely approaches the validation accuracy to around 85%. This implies no sign of overfitting and the model learning effectively. The model loss for the other visualised on the right and easily decreases with the training. Training and validation losses decline sharply in the beginning and level off close to a low value by epoch 50. The closeness of the model's train and validation metrics indicates that it can generalise well to the unseen data. The visualisations of these are critical for model learning behaviour and spotting a problem like underfitting or overfitting.



```
Visualizing predictions...
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 461ms/step
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras fo
Model saved as 'cifar10_cnn_model.h5'
```

Figure 7: Visualizing Predictions

Figure 7 shows the sample of the predictions made by the trained CNN model on CIFAR-10 test dataset. Each image includes a predicted label (e.g., "Pred: cat") and the corresponding ground truth (e.g., "True: cat"). The visualization highlights that the model accurately classifies most images, particularly those representing distinct classes like "cat," "airplane," and "frog." However, there are a few mismatches, such as images predicted as "automobile" but labeled as "cat" suggesting some confusion in visually similar classes. This type of prediction visualisation is important for evaluation beyond numerical accuracy, as it enables an understanding of how the model interprets various features. Additionally, this could be used to help in detecting the pattern in misclassification and directing further refinemodel'spracticalapplication is demonstrated by the predictions and is of intrinsic value for a qualitative performance assessment.

5. Discussion

A high level of effectiveness of the CNN model on the image classification tasks is shown in the CIFAR-10 dataset experimental results. The model has achieved test accuracy of 84.89%, which demonstrates strong generalisation to unseen data. Both training and validation accuracies increased consistently over the epochs, as is shown in Figure 6, and loss values were decreasing over the epochs, meaning that we could learn safely without overfitting too much. The model has high robustness of training and validation performance in a narrow gap. Also, Figure 7 gives qualitative validation with correct predictions for most categories. Misclassifications, especially between alike-looking objects such as cats and automobiles, are demonstrated to require better feature extraction. A few fluctuations in validation loss could have been due to data variability, class imbalance, etc in the training history as well. However, the CNN architecture turned out to be appropriate for recognising image tasks.

6. Conclusion and Recommendation

Finally, the results achieved an accuracy of 84.89%, i.e., 84.89% accuracy on our test set classifying images from CIFAR10. The training went smoothly and effectively, with the accuracy and loss decreasing at a constant rate. The prediction visualisations also helped confirm that the model could do well in recognising most image classes. Such outcomes further support the efficiency of deep learning architectures in particular, computer vision, i.e., CNNs. Nevertheless, there is some misclassification indicating the need for improvement. To improve generalisation and solve the problem of class similarity, you should attempt to experiment with deeper architectures such as ResNet or VGG or use data augmentation techniques. However, introducing regularisation methods, such as dropout, and batch normalisation, might enhance the performance even more. Deploying the model with a lightweight inference engine (an example being TensorFlow Lite) is practical if it leads to use of the model in real-time on edge devices. Future work will include hyperparameter tuning and ensembling to further enhance accuracy.

Author Contributions

The author has reviewed and approved the final manuscript with equal contribution.

Funding

External funding was not provided for this study.

Conflicts of Interest

The authors state that there are no conflicts of interest related to this paper.

References

[1] Akkem, Y., Biswas, S.K. and Varanasi, A., 2024. A comprehensive review of synthetic data generation in smart farming by using variationalautoencoder and generative adversarial network. Engineering Applications of Artificial Intelligence, 131, p.107881.

[2] Girin, L., Leglaive, S., Bie, X., Diard, J., Hueber, T. and Alameda-Pineda, X., 2020. Dynamical variationalautoencoders: A comprehensive review. arXiv preprint arXiv:2008.12595.

[3] Mak, H.W.L., Han, R. and Yin, H.H., 2023. Application of variationalautoEncoder (VAE) model and image processing approaches in game design. Sensors, 23(7), p.3457.

[4] Vahdat, A. and Kautz, J., 2020. NVAE: A deep hierarchical variationalautoencoder. Advances in neural information processing systems, 33, pp.19667-19679. Mishra, A., Krishna Reddy, S., Mittal, A. and Murthy, [5] H.A., 2018. A generative model for zero shot learning using conditional variationalautoencoders. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops (pp. 2188-2196).

Received: 2025-04-16
Accepted: 2025-05-18
Published Online: 2025-07-30
DOI: 10.70454/JRICST.2025.20301

[6] Innocent, E.K., 2024. Enhancing Data Security in Healthcare with Synthetic Data Generation: An Autoencoder and VariationalAutoencoder Approach (Master's thesis, Oslo Metropolitan University).

[7] Bengesi, S., El-Sayed, H., Sarker, M.K., Houkpati, Y., Irungu, J. and Oladunni, T., 2024. Advancements in Generative AI: A Comprehensive Review of GANs, GPT, Autoencoders, Diffusion Model, and Transformers. IEEe Access.

[8] Elbattah, M.; Loughnane, C.; Guérin, J.-L.; Carette, R.; Cilia, F.; Dequen, G. VariationalAutoencoder for Image-Based Augmentation of Eye-Tracking Data. J. Imaging 2021, 7, 83. https://doi.org/10.3390/jimaging7050083

[9] Di Fan, Y.W., Liu, H., Chen, Y. and Wei, A., Design and Implementation of Unity3D-based Image Compression Coding Gamification Teaching System.

[10] Şen, D., Küçükkaykı, H.T. and Sürer, E., 2021. Automated game mechanics and aesthetics generation using neural style transfer in 2d video games. BilişimTeknolojileriDergisi, 14(3), pp.287-300.

[11] Gan, Z., Chai, X., Zhang, J., Zhang, Y. and Chen, Y., 2020. An effective image compression–encryption scheme based on compressive sensing (CS) and game of life (GOL). Neural Computing and Applications, 32, pp.14113-14141.

[12] Xu, S., Guo, C., Zhu, Y., Liu, G. and Xiong, N., 2023. CNN-VAE: An intelligent text representation algorithm. The Journal of Supercomputing, 79(11), pp.12266-12291.

[13] Akbari, M. and Liang, J., 2018, April. Semi-recurrent CNN-based VAE-GAN for sequential data generation. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 2321-2325). IEEE.

[14] Metlapalli, A.C., Muthusamy, T. and Battula, B.P., 2020. Classification of Social Media Text Spam Using VAE-CNN and LSTM Model. Ingénierie des Systèmes d Inf., 25(6), pp.747-753.

[15] Liu, J., Yang, G., Li, X., Hao, S., Guan, Y. and Li, Y., 2022. A deep generative model based on CNN-CVAE for wind turbine condition monitoring. Measurement Science and Technology, 34(3), p.035902.

[16] Kim, J.Y. and Cho, S.B., 2021. Deep CNN transferred from VAE and GAN for classifying irritating noise in automobile. Neurocomputing, 452, pp.395-403.