CHAPTER II

LITERATURE REVIEW

2.1 Related Work

Data enhancement has emerged as a vital technique in the creation of artificial intelligence models. A leading strategy involves the use of Generative Adversarial Networks (GANs) for augmentation, capable of producing artificial data that mimics the original closely. This technique is not restricted to standard images but also applies to various fields.

a. The study conducted by Huang et al. (2022) demonstrates that data augmentation can enhance model accuracy in detecting diseases in sugarcane plants. In their research, the use of DCGAN significantly improved the accuracy of the MobileNetV3-Large model, increasing it from 53.5% to 99% following background removal and the addition of synthetic data generated by DCGAN. These findings suggest that data augmentation not only increases the volume of training data but also introduces variability that enables the model to recognize more complex patterns. The study utilized two distinct datasets, comprising 790 images for segmentation training and 120 images from another dataset for classification training. By leveraging DCGAN to generate synthetic data as part of the data augmentation process, the existing dataset can be expanded, and image diversity can be enhanced, ultimately potentially improving the performance of the classification model post-segmentation.
b. Zhang et al. (2021) shown significant potential in image processing using DCGAN. DCGAN was used to augment hard-to-obtain defective pear images,

c. improving the accuracy of the CNN model to 97.35% on the validation data.

d. The study conducted by Wu et al. (2020) demonstrated that combining 1,500 original images with 3,800 augmented images of tomato leaf diseases resulted in a GoogLeNet model achieving an average top-1 identification accuracy of 94.33%. This approach not only increased the dataset size and diversity but also enhanced the model's generalization capability.

e. The study conducted by Liu et al. (2021) demonstrated that DCGAN can generate 2.02 times better images with 1.55 times higher diversity than traditional GAN models on MNIST dataset.

f. The research conducted by Bird and colleagues (2022) revealed that incorporating synthetic data greatly enhances the precision of the VGG16 model. This investigation utilized the SoftwareMill dataset, which contains 2,690 images of lemons, each with a resolution of 256×256 pixels, divided into two categories: healthy and unhealthy. In the absence of data augmentation, the model reached an accuracy of 83.77%, while integrating 400 pieces of synthetic data boosted the accuracy to 88.75%. Furthermore, the research emphasized that the correct quantity of synthetic data is vital for improving the performance of the model.

g. The research carried out by Thaniket and colleagues in 2025 focused on creating an automated classification system for bird-of-paradise species utilizing a CNN framework, particularly MobileNetV2. The findings indicated that the model attained a training accuracy of 98.49% and a validation accuracy of 97.50%. Assessment through a confusion matrix showed a significant degree of accuracy with few misclassification errors, underscoring the model's

capability in correctly identifying the species. The dataset employed in this research comprised 435 images.

h. The research carried out by Azahro Choirunisa and others in 2021 focused on identifying and categorizing cat breeds employing a CNN approach utilizing the EfficientNet-B0 structure. The best-performing model attained an accuracy of 95% during tests conducted on a collection of 180 cat photographs. The dataset utilized in this research comprised a total of 2700 images.

2.2 Theoritical review

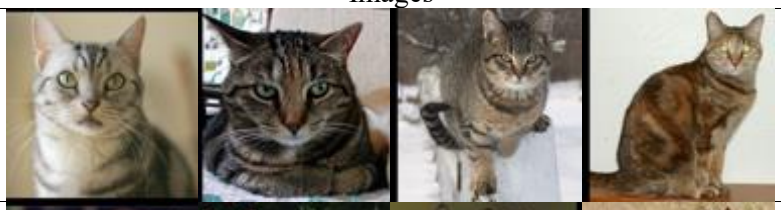
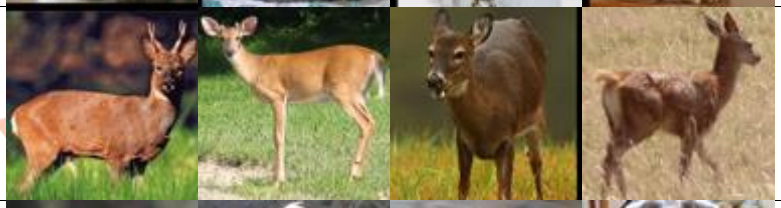
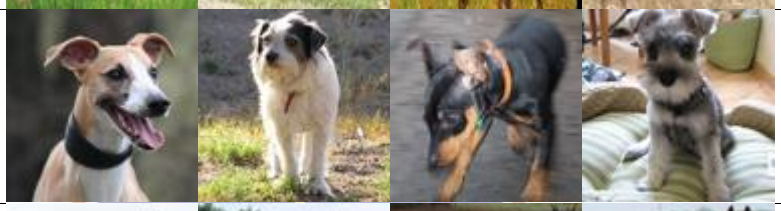
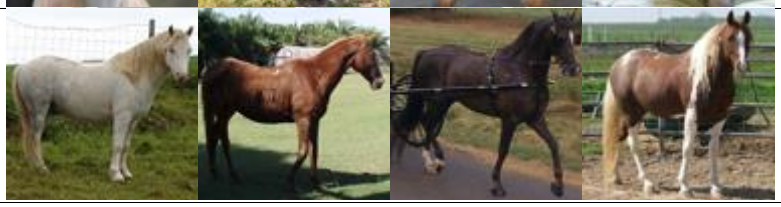
This portion outlines the theoretical principles that underpin this research's development. It includes various important ideas such as the STL-10 dataset, Generative Adversarial Networks (GAN), along with its variation Deep Convolutional Generative Adversarial Network (DCGAN), and essential deep learning principles including Convolutional Neural Networks (CNN), EfficientNet-B0, and MobileNetV2. Furthermore, this section examines the evaluation methodologies utilized in this research, such as the confusion matrix, Fréchet Inception Distance (FID), and Inception Score (IS), which are used to evaluate the effectiveness of both the generative and classification models.

2.2.1 STL-10

STL-10 is a collection of pictures made to help teach deep learning models without needing supervision. The dataset is similar to CIFAR-10 in terms of class composition, but each class in the STL-10 training data contains fewer labeled examples compared to CIFAR-10. However, STL-10 compensates for this with a significantly larger set of unlabeled images, which can be utilized to train image

models in an unsupervised fashion for generating self-trained weights. Unlike CIFAR-10, which contains images with a resolution of 32 × 32 pixels, STL-10 offers higher-resolution images at 96 × 96 pixels. The dataset includes 1.300 images each class (Papastratis, 2021). The example of the stl-10 dataset can be seen in Table 2.1.

Table 2.1
Sample Images of STL-10 Dataset

Class	Images
Cat	
Deer	
Dog	
Horse	

As shown in Table 2.1, the STL-10 dataset used in this study consists of four selected classes: cat, deer, dog, and horse. Each row in the table presents several sample images representing the visual characteristics of each class. The images illustrate the diversity of object poses, backgrounds, lighting conditions, and viewpoints present in the dataset. This variability is important for training deep

learning models, as it helps the models learn more robust and generalized visual features for classification tasks.

2.2.2 Generative Adversarial Network (GAN)

Generative Adversarial Network, or GAN, is a method in deep learning that creates new data similar to real data. A GAN has two parts that work against each other: the generator and the discriminator. The generator's job is to make fake data, while the discriminator's job is to tell apart real data from the data made by the generator. These two parts are trained at the same time in a competitive way. The generator keeps getting better at fooling the discriminator, and the discriminator improves its skills in recognizing the differences between real and fake data. Because of this ongoing process, GANs can create data that looks a lot like the original data set. An image showing how GAN works can be found in Figure 2.1.

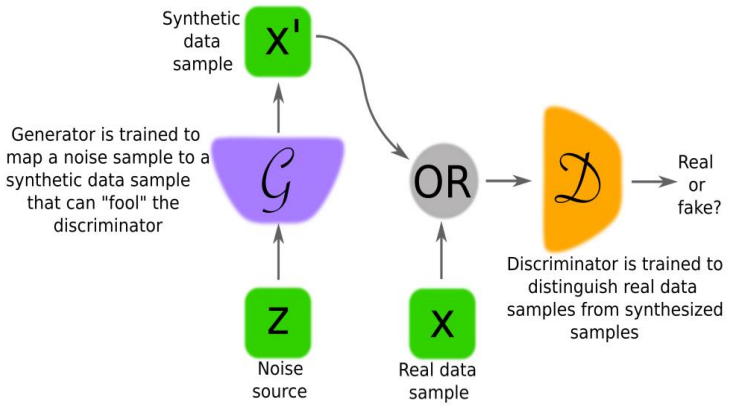


Figure 2.1 How GAN Works
Source: (Creswell et al., 2018)

As shown in Figure 2.1, the GAN system is made up of two neural networks that work together in a competitive way: the Generator (G) and the Discriminator

(D). The generator takes a random noise vector (z) and changes it into fake data that looks like the real examples from the dataset. On the other hand, the discriminator gets both real data and fake data and tries to tell them apart by guessing if the input is real or fake. During training, the generator keeps getting better at creating realistic data to trick the discriminator, while the discriminator also gets better at telling the difference between real and fake samples. Through this back-and-forth learning, the generator learns more about the real data patterns and starts to create even more believable fake data.

2.2.3 Deep Convolutional Generative Adversarial Network (DCGAN)

The Deep Convolutional Generative Adversarial Network, or DCGAN, is a more advanced version of the regular Generative Adversarial Network, also known as GAN. It uses convolutional neural networks, or CNNs, in both the parts that create images and the parts that judge them. This change makes it better at making clear and detailed images. DCGAN fixes some major problems that the original GAN had, especially when it came to how reliably it could be trained and how realistic the images looked.

In DCGAN, the part that generates images starts with something called a latent vector. This vector is usually taken from a type of randomness, like Gaussian or uniform distribution, and it acts as a small summary of what the final image should look like. This vector gets turned into a fake image through several layers of transposed convolutional layers, which are also called deconvolutional layers. Each of these layers makes the image bigger and sharper while improving the features. Batch normalization helps keep the range of outputs consistent, and ReLU

activation functions add some non-linearity, which helps with how the information moves through the network.

On the other hand, the discriminator is a type of binary classifier based on Convolutional Neural Networks (CNNs). It gets two kinds of information: real images from the dataset or fake images produced by the generator. It processes these images through several layers that gradually make the images smaller while identifying important features within them. To help with this, leaky ReLU activation functions are often used in the discriminator. This lets a small amount of negative input get through, which stops the neurons from shutting down and makes it easier for gradients to flow.

The way it learns is based on a competitive idea: the generator's job is to make pictures that look as real as possible, while the discriminator's job is to tell the difference between real and fake images. This back-and-forth creates a minimax optimization problem, where both systems keep getting better by reacting to what the other is doing. Over many training cycles, the generator starts to understand the patterns in the real data, allowing it to create fake images that look very similar to real ones. You can see how the DCGAN is set up in Figure 2.2.

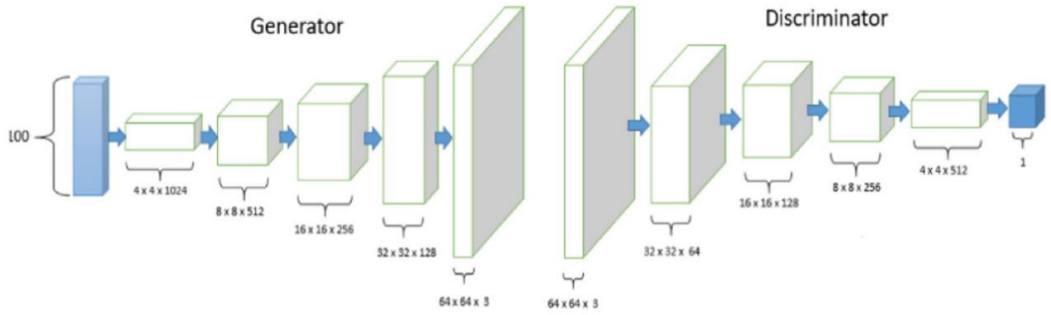


Figure 2.2 DCGAN Architecture
Source: (Jenkins et al., 2024)

As shown in Figure 2.2, the DCGAN model has two key parts: the generator and the discriminator. The generator takes in a noise vector that is 100 units long and changes it step by step into a fake image using several layers that work in reverse. These layers make the image larger and clearer, beginning with a simple version and ending with an image that is 64 by 64 pixels and has three color channels (RGB). To help with training and make the information flow better, batch normalization and ReLU activation functions are used in the middle layers. In the last layer, a Tanh activation function is used to create pixel values that range from -1 to 1.

2.2.4 Deep Learning

Deep learning is a type of machine learning that uses complex layers of artificial neural networks to automatically understand and learn from data. It is capable of identifying non-linear patterns across spatial and temporal dimensions without requiring prior transformations on the input data (Solórzano et al., 2021).

This concept mimics the way humans learn, where the model acquires knowledge from example data without needing explicit programming for each individual feature. Deep learning is used a lot in different areas like recognizing objects, understanding human language, recognizing speech, analyzing feelings, and creating images. Neural networks are made up of connected points that work to manage and keep information (Ichwan & Hadi, 2023). This approach really boosts how quickly and effectively we can train these models, allowing them to work with big amounts of data accurately. You can see how a neural network is structured in Figure 2.3.

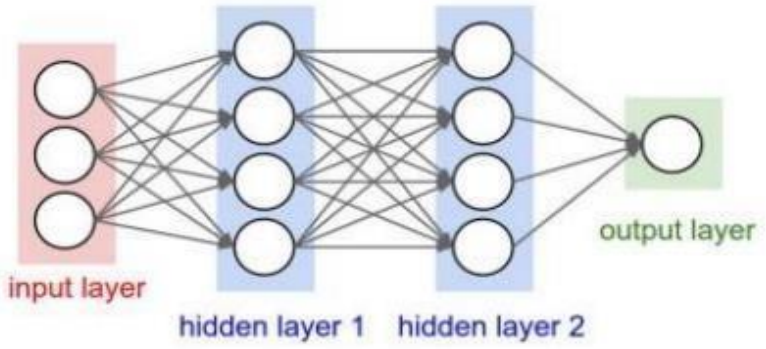


Figure 2.3 Neural Network Architecture
Source: (Ichwan & Hadi, 2023).

As illustrated in Figure 2.3, a neural network has an input layer, one or more hidden layers, and an output layer. The input layer takes in data, and then it goes through hidden layers where neurons change the information using weights and activation functions. The output layer creates the final result, which includes predictions or classifications based on the patterns it has learned.

2.2.5 Convolutional Neural Network (CNN)

A Convolutional Neural Network, or CNN, is a special kind of artificial neural network made to work with data that is arranged in a grid, like digital pictures. The structure of a CNN usually has different types of layers: convolutional layers, pooling layers, and fully connected layers. Convolutional layers help find important details in the input image by using filters that move across the picture. Pooling layers help make the data simpler, which boosts speed and reduces the chance of mistakes by not fitting perfectly to the training data. Fully connected layers take the features that were identified and combine them into a form that can be used for tasks like identifying or classifying things. By working together, these layers allow CNNs to find patterns in pictures very accurately. This technology is used in many areas,

such as recognizing faces, spotting objects, classifying plant diseases, and analyzing medical images, showing how well it can manage complicated visual information. The design of a CNN can be illustrated in Figure 2.4.

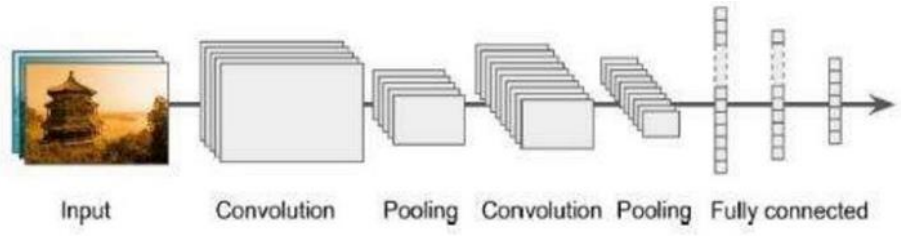


Figure 2.4 CNN Architecture
Source: (Hariz, Yulita, & Suryana, 2022)

As illustrated in Figure 2.4, a CNN handles an image by going through different steps, which are convolution, pooling, and fully connected layers. The convolutional layers take out key visual details from the image, and the pooling layers make the size smaller to make the calculations faster. After that, the features that have been taken out move on to the fully connected layers to produce the final result of the classification.

2.2.6 EfficientNet-B0

EfficientNet-B0 is a CNN optimized using the compound scaling technique, which balances model size, input resolution, and network depth for efficient performance. It processes input images through multiple convolutional layers while adapting model size and depth to extract features with fewer parameters than other CNNs, without compromising accuracy. This makes EfficientNet-B0 highly effective for image classification, even with limited data. The architecture of EfficientNet-B0 can be seen in Figure 2.5.

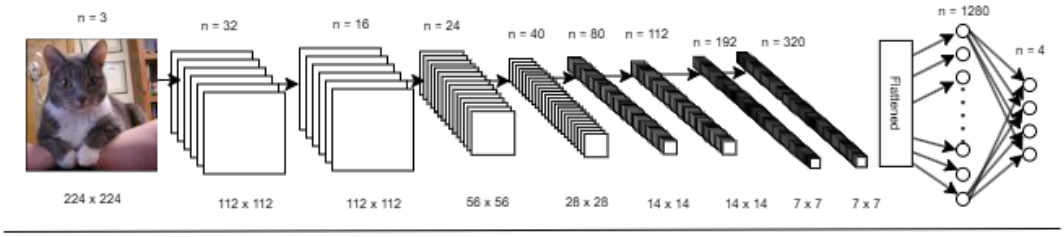


Figure 2.5 EfficientNet-B0 Architecture

As shown in Figure 2.5, the EfficientNet-B0 design takes an input image and runs it through a series of convolutional layers that help to gradually grab important visual details. In the beginning layers, the model focuses on picking up simple features like edges and textures, while the layers that come after start to recognize more complicated shapes and patterns of objects. A unique part of EfficientNet-B0 is its method of compound scaling, which increases the network's depth, width, and input size all at the same time in a balanced way. Once the features have been extracted, these detailed feature maps are then turned into a flat format and sent through fully connected layers to create the final output for classification.

2.2.7 MobileNetV2

MobileNetV2 is a type of CNN that is designed to be efficient by using inverted residuals and linear bottlenecks, as shown in figure 3. It uses depthwise separable convolutions to lessen the number of parameters and make the process faster while still performing well, which is perfect for devices with limited resources. The model takes in images that are 224x224 pixels through an initial convolution layer, then goes through inverted residual blocks that make the features bigger, perform depthwise convolutions, and bring back the channel size with a linear bottleneck. This setup makes sure there's a good mix of accuracy and speed

for tasks like classifying images and detecting objects. You can see the structure of MobileNetV2 in Figure 2.6.

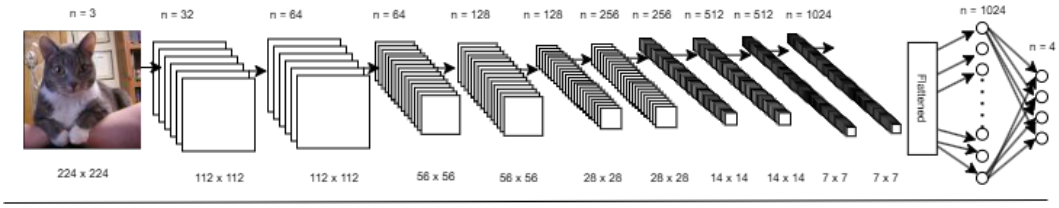


Figure 2.6 MobileNetV2 Architecture

As illustrated in Figure 2.6, the MobileNetV2 architecture processes input images through a sequence of convolutional layers and inverted residual blocks. The model begins with an initial convolution layer that extracts basic visual features from the input image. These features are then passed through several inverted residual blocks, where the feature dimensions are first expanded using pointwise convolution, followed by depthwise convolution to capture spatial information, and finally compressed back to a lower-dimensional representation using a linear bottleneck. This structure enables MobileNetV2 to efficiently learn meaningful feature representations while maintaining a relatively small number of parameters. After the feature extraction process, the resulting feature maps are flattened and processed by fully connected layers to produce the final classification output.

2.2.8 Confusion Matrix

The Confusion Matrix is a chart that helps show how well a model works when sorting data (Normawati et al., 2021). This chart shows how many predictions the model got right and wrong by comparing them to the real answers. The matrix has different parts, including True Positives (TP), True Negatives (TN), False

Positives (FP), and False Negatives (FN), which all give a complete picture of how well the model is classifying things. More information about the confusion matrix is shown in Figure 2.7.

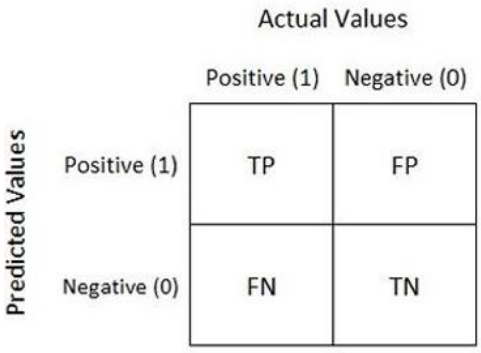


Figure 2.7 Confusion Matrix
Source: (Atmajaya, Febrianti, & Darwis, 2023)

In Figure 2.7, the confusion matrix is presented to evaluate the performance of the model, comprising several components:

a. True Positives (TP): The number of correct predictions where the model predicts a positive class and the actual value is also positive.
b. True Negatives (TN): The number of correct predictions where the model predicts a negative class and the actual value is also negative.
c. False Positives (FP): The number of incorrect predictions where the model predicts a positive class, but the actual value is negative.
d. False Negatives (FN): The number of incorrect predictions where the model predicts a negative class, but the actual value is positive.

The confusion matrix enables the computation of various evaluation metrics that are essential for assessing the performance of a classification model, including:

a. Accuracy, defined as the proportion of total correct predictions, calculated by dividing the number of correct predictions by the total number of predictions.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (1)$$

b. Precision, the proportion of correctly predicted positive observations, calculated by dividing the number of true positive predictions by the total number of positive predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2)$$

c. Recall (Sensitivity or True Positive Rate), the proportion of actual positive instances that are correctly identified by the model. Positive instances are calculated as follows.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (3)$$

d. F1-score, the harmonic mean of precision and recall, combining both metrics into a single value. The F1-score can be calculated as follows.

$$\text{F1} - \text{score} = 2\text{x}\frac{\text{Precision x Recall}}{\text{Precision} + \text{Recall}} \quad (4)$$

2.2.9 Fréchet Inception Distance

Fréchet Inception Distance, or FID, is a popular way to check how good and similar images created by AI are compared to real images. FID works by finding the difference between two sets of features, one from real images and one from the AI-generated images. These features come from a specific layer in the Inception-v3 neural network, which creates a set of 2048 number values to describe the

images. In simple terms, FID looks at the Fréchet distance between two groups of data that follow a multivariate Gaussian pattern, which represents these features. When the FID score is lower, it means that the images made by the AI look more like the real ones in a statistical sense. The way to calculate Fréchet Inception Distance is shown in Equation (5).

$$\text{FID} = ||\pi_{\text{real}} - \pi_{\text{fake}}||^2 + \text{Tr}(\sum_{\text{real}} + \sum_{\text{fake}} - 2(\sum_{\text{real}}\sum_{\text{fake}})^{1/2}) \quad (5)$$

a. π_{real} = The mean vector of the feature representations extracted from real data s. It represents the center (expected value) of the real data distribution in feature space.
b. π_{fake} = The mean vector of the feature representations extracted from generated (synthetic) images. It represents the center of the generated data distribution in feature space.
c. $\sum_{\text{real}}$ = The covariance matrix of the feature representations of real data s. It captures how the features of real data vary and co-vary with each other.
d. $\sum_{\text{fake}}$ = The covariance matrix of the feature representations of generated images. It captures the variation and correlation between features of synthetic data.
e. Tr( ) = The trace operator, which returns the sum of the diagonal elements of a matrix. In this context, it sums the total variances across all feature dimensions.

2.2.10 Inception Score

Inception Score (IS) is a way to measure how good and varied the pictures created by generative models, like Generative Adversarial Networks (GANs), are. IS uses a model called Inception-v3, which has already been trained, to check how sharp and different the pictures are. Basically, IS figures out the average difference between two distributions: one that is based on the labels of the pictures given their features and another that just looks at the overall distribution of labels. Good quality pictures should have a clear distribution of labels (meaning they fit well into one category) and also show a wide range of different types across all the pictures created. A higher IS means the pictures are realistic (easily placed into categories) and varied (representing many categories) (Salimans et al., 2016). The Inception Score is described in Equation (6).

$$\mathrm{IS} = \exp\left(\mathrm{E}_{x\sim pg}[\mathrm{D}_{\mathrm{KL}}(\mathrm{P}(y|x) \,||\, \mathrm{P}(y))]\right) \quad (6)$$

a. $x\sim pg(x)$ = Indicates that the input image x is sampled from the generator's distribution, which represents the set of generated images.
b. $P(y|x)$ = The conditional class probability distribution predicted by the Inception model given the image x. It shows how confidently the model classifies the generated image into one of the known categories.
c. $P(y)$ = The marginal class distribution, computed as the average of $p(y|x)$ over all generated images. It represents the overall diversity of the class predictions in the generated dataset.
d. $\mathrm{D}_{\mathrm{KL}}(P(y|x) \,||\, P(y))$ = The Kullback-Leibler (KL) divergence between the conditional and marginal distributions. This measures how much the class

prediction for a single image deviates from the average prediction across all images.

e. $E_x[]$ = The expectation operator that averages the KL divergence over all generated samples.

f. exp () = The exponential function is applied to ensure that the score is strictly positive and interpretable. A higher score indicates both high image quality and high class diversity.