

Cifar-10 Image Generation GAN

Name: Soh Hong Yu

Admin Number: P2100775

Class: DAAA/FT/2B/01

Module Code: ST1504 Deep Learning

References (In Harvard format):

1. Goodfellow, I.J. et al. (2014) Generative Adversarial Networks, arXiv.org.
Available at: <https://arxiv.org/abs/1406.2661> (Accessed: January 21, 2023).
2. Cox, S. (2021) The overlooked technique of image averaging, Photography Life.
Available at: <https://photographylife.com/image-averaging-technique> (Accessed: November 19, 2022).
3. Brownlee, J. (2019) How to evaluate generative Adversarial Networks, MachineLearningMastery.com.
Available at: <https://machinelearningmastery.com/how-to-evaluate-generative-adversarial-networks/> (Accessed: January 21, 2023).
4. Hui, J. (2020) Gan - ways to improve gan performance, Medium. Towards Data Science.
Available at: <https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b> (Accessed: January 21, 2023).
5. Briggs, J. (2020) Beating the gan game, Medium. Towards Data Science.
Available at: <https://towardsdatascience.com/beating-the-gan-game-aefcce0a20be> (Accessed: January 21, 2023).
6. Silva, T.S. (2017) A short introduction to generative Adversarial Networks, A Short Introduction to Generative Adversarial Networks - Thalles' blog.
Available at: <https://sthalles.github.io/intro-to-gans/> (Accessed: January 21, 2023).
7. A. Radford, L. Metz, en S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", arXiv [cs.LG]. 2016.
8. T. Miyato, T. Kataoka, M. Koyama, en Y. Yoshida, "Spectral Normalization for Generative Adversarial Networks", arXiv [cs.LG]. 2018.
9. Hui, J. (2021) Gan - spectral normalization, Medium. Medium.
Available at: <https://jonathan-hui.medium.com/gan-spectral-normalization-893b6a4e8f53> (Accessed: January 21, 2023).
10. Salimans, T. and Kingma, D.P. (2016) Weight normalization: A simple reparameterization to accelerate training of deep neural networks, arXiv.org.
Available at: <https://arxiv.org/abs/1602.07868> (Accessed: January 21, 2023).
11. Tensorflow (2023) Customize what happens in model.fit : Tensorflow Core, TensorFlow.
Available at:
https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit#wrapping_up_an_end-to-end_gan_example (Accessed: January 21, 2023).
12. Google (2023) Common problems | machine learning | google developers, Google. Google.
Available at: <https://developers.google.com/machine-learning/gan/problems> (Accessed: January 21, 2023).

13. Hui, J. (2019) Gan - why it is so hard to train generative adversarial networks!, Medium. Medium.
Available at: <https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b> (Accessed: January 21, 2023).
14. Lazarou, C. (2020) Why do gans need so much noise?, Medium. Towards Data Science.
Available at: <https://towardsdatascience.com/why-do-gans-need-so-much-noise-1eae6c0fb177> (Accessed: January 21, 2023).
15. Brownlee, J. (2019) How to calculate the KL divergence for Machine Learning, MachineLearningMastery.com.
Available at: <https://machinelearningmastery.com/divergence-between-probability-distributions/> (Accessed: January 21, 2023).
16. Agrawal, T. (2022) Gans failure modes: How to identify and monitor them, neptune.ai.
Available at: <https://neptune.ai/blog/gan-failure-modes> (Accessed: January 21, 2023).
17. Brownlee, J. (2019) How to implement the Frechet Inception Distance (FID) for evaluating Gans, MachineLearningMastery.com.
Available at: <https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/> (Accessed: January 21, 2023).
18. Papers with code (2023) Papers with code - leaky relu explained, Explained | Papers With Code.
Available at: <https://paperswithcode.com/method/leaky-relu> (Accessed: January 21, 2023).
19. Papers with code (2023) Papers with code - prelu explained, PReLU Explained | Papers With Code.
Available at: <https://paperswithcode.com/method/prelu> (Accessed: January 21, 2023).
20. Mishra, D. (2021) Transposed convolution demystified, Medium. Towards Data Science.
Available at: <https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba> (Accessed: January 21, 2023).
21. Thakur, A. (2021) How to evaluate Gans using Frechet Inception Distance (FID), W&B.
Available at: <https://wandb.ai/ayush-thakur/gan-evaluation/reports/How-to-Evaluate-GANs-using-Frechet-Inception-Distance-FID---Vmldzo0MTAxOTI> (Accessed: January 21, 2023).
22. Viswanath, P. (2017) 12.1 generative adversarial network - courses.engr.illinois.edu, ECE598: Representation Learning: Algorithms and Models.
Available at: https://courses.engr.illinois.edu/ece598pv/fa2017/Lecture12_GAN_AmirT.pdf (Accessed: January 20, 2023).
23. Mescheder, L., Geiger, A. and Nowozin, S. (2018) Which training methods for gans do actually converge?, arXiv.org.
Available at: <https://arxiv.org/abs/1801.04406> (Accessed: January 21, 2023).
24. Kodali, N. et al. (2017) On convergence and stability of gans, arXiv.org.
Available at: <https://arxiv.org/abs/1705.07215> (Accessed: January 21, 2023).
25. Papers with code (2023) Papers with code - CIFAR-10 benchmark (image generation), The latest in Machine Learning.
Available at: <https://paperswithcode.com/sota/image-generation-on-cifar-10> (Accessed: January 29, 2023).
26. Jung, S. and Keuper, M. (2021) Internalized biases in Fréchet inception distance, OpenReview.
Available at: <https://openreview.net/forum?id=mLG96UpmbYz> (Accessed: January 29, 2023).

Project Objective

Implement a suitable GAN architecture to the problem. Creating 1000 new images by training the model on the cifar-10 dataset.

Background Information

In the world where data is very important, there are certain information that is sensitive or private. Using GAN allow us to generate new and unseen images which can be used for training of other models etc.

CIFAR-10 dataset (Canadian Institute for Advanced Research) is a collection of images that are commonly used to train and benchmark image classification algorithms. It is a subset of 80 million tiny images dataset and consists of 60,000 32 x 32 color images in 10 different classes. There are 50000 training images and 10000 test images. It was collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. [1]

Initialising Libraries and Variables

```
In [ ]: import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras.backend as K
from tensorflow import keras
from keras.layers import Reshape, Conv2DTranspose, PReLU
from keras.utils import to_categorical, plot_model
from keras.layers import Concatenate
import numpy as np
from keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.image import resize
from scipy.linalg import sqrtm
import math
from tqdm.notebook import tqdm
import tensorflow as tf
from IPython.display import clear_output, HTML
import imageio
import glob
from keras.layers import AveragePooling2D, ZeroPadding2D, BatchNormalization, Activation, Max
from keras.layers import Normalization, Dense, Conv2D, Dropout, BatchNormalization, ReLU
from keras.models import Sequential, Model
from keras import Input
from keras.optimizers import *
from keras.callbacks import EarlyStopping
import visualkeras
from keras.initializers import RandomNormal
from tensorflow_addons.layers import SpectralNormalization
from keras.layers import LeakyReLU, GlobalMaxPooling2D, GlobalAveragePooling2D
```

```
In [ ]: # Fix random seed for reproducibility
seed = 88
np.random.seed(seed)
```

Checking GPU

```
In [ ]: # Check if Cuda GPU is available
tf.config.list_physical_devices('GPU')
```

```
Out[ ]: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

Loading Datasets

```
In [ ]: df = tf.keras.datasets.cifar10.load_data()
```

```
In [ ]: (x_train, y_train), (x_test, y_test) = df
```

Exploratory Data Analysis

We will begin by conducting an exploratory data analysis of the data, to gain a better understanding of the characteristics of the dataset.

x_train: uint8 NumPy array of coloured image data with shapes (50000, 32, 32, 3), containing the training data. Pixel values range from 0 to 255.

y_train: uint8 NumPy array of labels (integers in range 0-9) with shape (50000, 1) for the training data.

x_test: uint8 NumPy array of coloured image data with shapes (10000, 32, 32, 3), containing the test data. Pixel values range from 0 to 255.

y_test: uint8 NumPy array of labels (integers in range 0-9) with shape (10000, 1) for the test data.

There are 10 different type of labels in the dataset. From the dataset, each value represent an item. The following list is the description of each value.

Item Labels

- 0 : Airplane
- 1 : Automobile
- 2 : Bird
- 3 : Cat
- 4 : Deer
- 5 : Dog
- 6 : Frog
- 7 : Horse
- 8 : Ship
- 9 : Truck

```
In [ ]: class_labels = {  
    0: 'Airplane',  
    1: 'Automobile',  
    2: 'Bird',  
    3: 'Cat',  
    4: 'Deer',  
    5: 'Dog',  
    6: 'Frog',  
    7: 'Horse',  
    8: 'Ship',  
    9: 'Truck'  
}  
  
NUM_CLASS = 10
```

Each image is a 32x32 image as well as 3 color channel [RGB] (coloured image). Therefore, we can set the IMG_SIZE as a tuple (32, 32, 3)

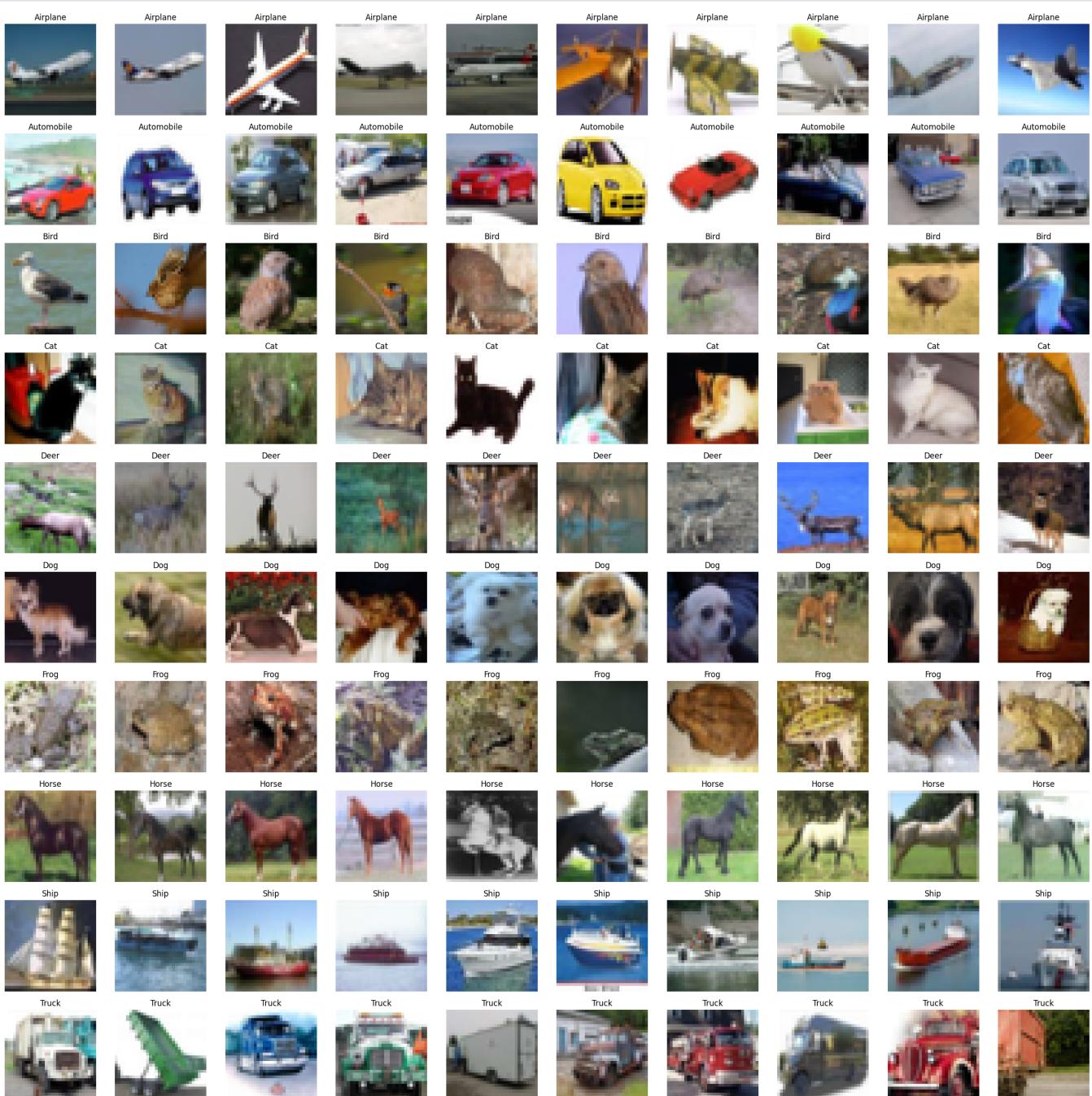
```
In [ ]: IMG_SIZE = (32, 32, 3)
```

Visualising the Dataset

Let's look at what the images look like.

```
In [ ]: fig, ax = plt.subplots(10, 10, figsize=(30, 30))
for i in range(10):
    images = x_train[np.squeeze(y_train == i)]
    random_index = np.random.choice(images.shape[0], 10, replace=False)
    images = images[random_index]
    label = class_labels[i]
    for j in range(10):
        subplot = ax[i, j]
        subplot.axis("off")
        subplot.imshow(images[j])
        subplot.set_title(label)

plt.show()
```



Observation

We notes that the images are correctly labelled. However, there are some concerns which the AI might use to help generate these images. For example, the cat image has text around the cat. This might be an issue as the AI might mistaken the cat and try to generate images of text inside which can affect our

desired results. As well as the different species of bird like an ostrich and sparrow. This might be an issue as the AI will not be able to clearly differentiate the different species and generate an amalgamation. Another issue with the data is the zoom of the images. The images have different sizing like for the dogs and cat. Sometimes the images are just the face of the animal or sometimes the images are a zoomed out version and the entire body of the animal can be seen.

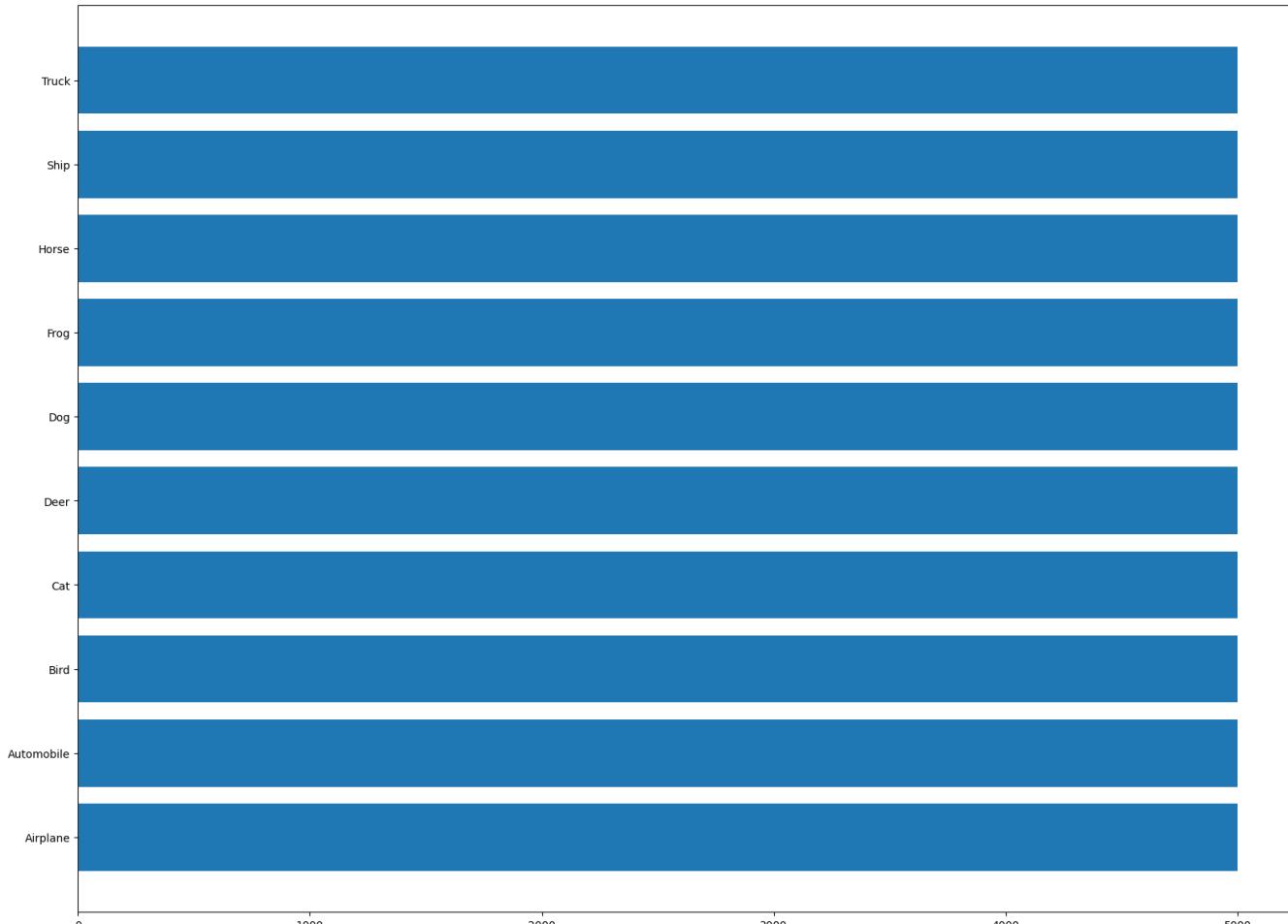
Class Distribution

When training a machine learning model, it is always important to check the distribution of the different classes in the dataset. This will inform us which metrics is the best to use and if anything is needed to balance the classes.

```
In [ ]: labels, counts = np.unique(y_train, return_counts=True)
for label, count in zip(labels, counts):
    print(f"{class_labels[label]": {count}}")
```

```
Airplane: 5000
Automobile: 5000
Bird: 5000
Cat: 5000
Deer: 5000
Dog: 5000
Frog: 5000
Horse: 5000
Ship: 5000
Truck: 5000
```

```
In [ ]: plt.figure(figsize=(20, 15))
plt.barh(labels, counts, tick_label=list(class_labels.values()))
plt.show()
```



Observations

As we can see from the bar graph, the distribution of the images is even. And there will be no bias towards one particular class

Image Pixel Distribution

We need to know the pixel intensity and know the distribution of the pixels

```
In [ ]: print("Max: ", np.max(x_train))
print("Min: ", np.min(x_train))
```

```
Max: 255
Min: 0
```

As expected, our pixels have values between 0 and 255.

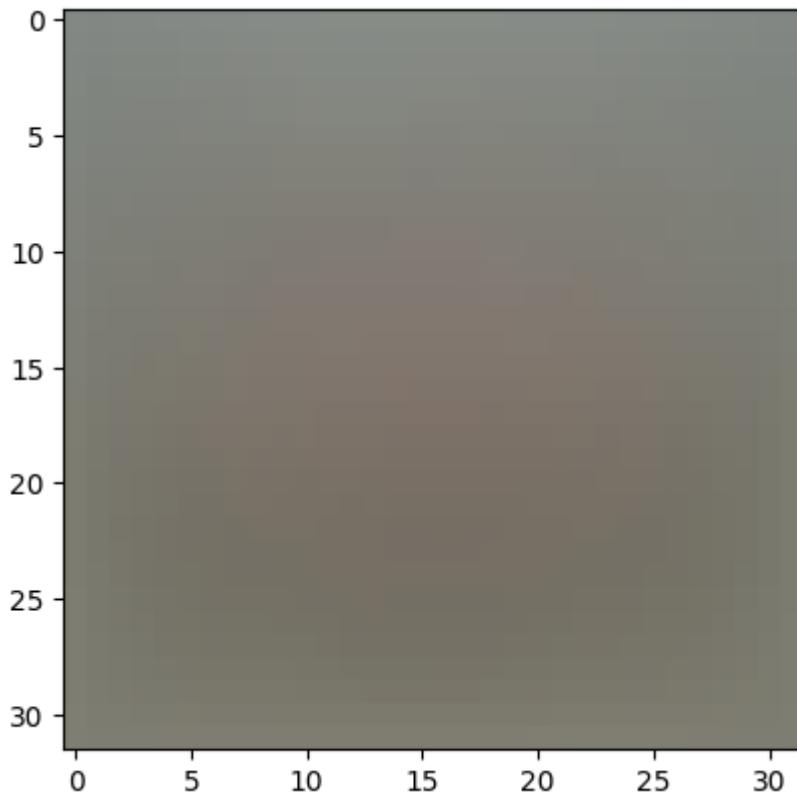
```
In [ ]: mean, std = np.mean(x_train, axis=(0, 1, 2)), np.std(x_train, axis=(0, 1, 2))
print("Mean:", mean)
print("std:", std)
```

```
Mean: [125.30691805 122.95039414 113.86538318]
std: [62.99321928 62.08870764 66.70489964]
```

Image Averaging

Image Averaging involves stacking multiple photos on top of each other and averaging them together. The main purpose is to see the noise of the image and therefore reducing it.

```
In [ ]: plt.imshow(np.mean(x_train, axis=0) / 255)
plt.show()
```

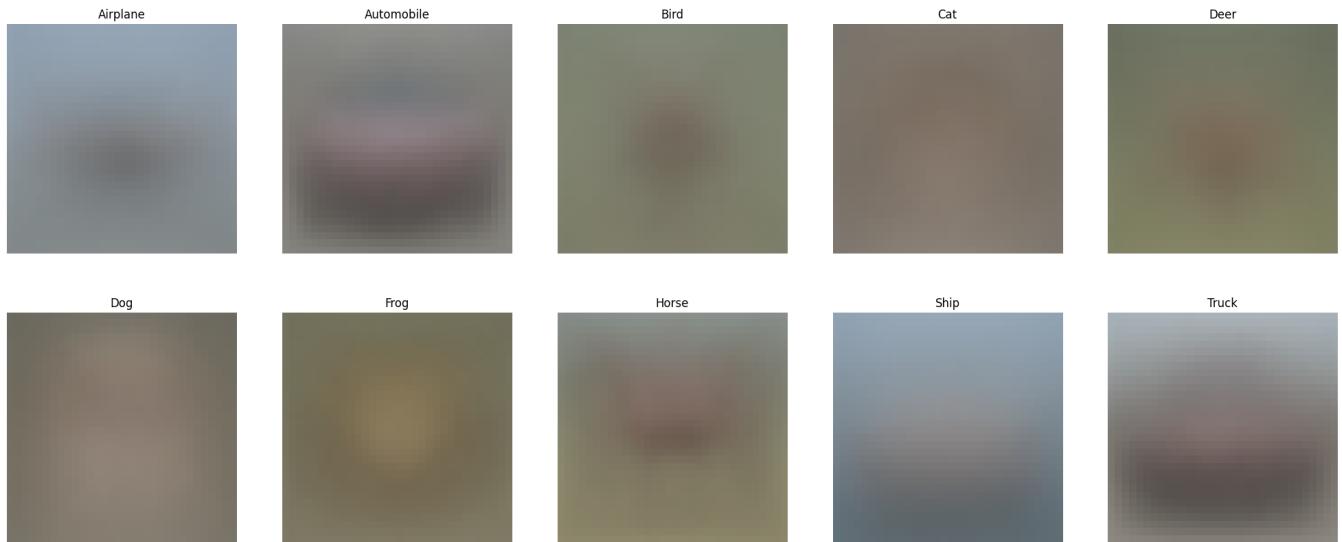


Observation

We cannot see a single thing from the image. This is likely due to the color of the images overlaying each other giving this blur effect.

```
In [ ]: fig, ax = plt.subplots(2, 5, figsize=(25, 10))

for idx, subplot in enumerate(ax.ravel()):
    avg_image = np.mean(x_train[np.squeeze(y_train == idx)], axis=0) / 255
    subplot.imshow(avg_image)
    subplot.set_title(f"class_labels[{idx}]")
    subplot.axis("off")
```



Observations

Although the average images is blurry, we can make out the images of an automobile, horse and truck etc. It is more difficult to make out the average image for the other classes. This suggests that for these similar class, the generated images might be able to learn more robust and representative features of the training data which means the images that will be generated will be very good.

Data Preprocessing

Before modelling, its is important to perform data preprocessing

One Hot Encoding

As they are, the current labels are encoded from 0-10, we will one hot encode the labels.

```
In [ ]: y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```
In [ ]: print(y_train[0])
print("Label:", tf.argmax(y_train[0]))
```

```
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
Label: tf.Tensor(6, shape=(), dtype=int64)
Label: tf.Tensor(6, shape=(), dtype=int64)
```

Normalizing the inputs

Image normalisation is done to the dataset.

Normalising the inputs means that we will calculate the mean and standard deviation of the training set, and then apply the formula below.

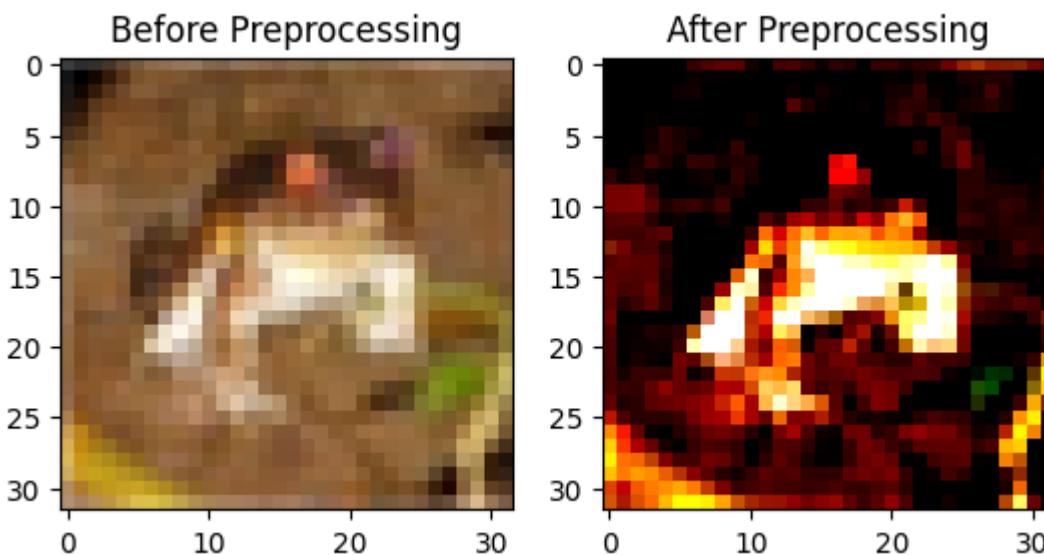
$$X = \frac{X - \mu}{\sigma}$$

Pixel values of each pixel are on similar scale, therefore normalisation can be used. This helps to optimize the algorithm to better converge during gradient descent.

```
In [ ]: pre_processing_v1 = Normalization()  
pre_processing_v1.adapt(x_train)
```

```
In [ ]: fig, ax = plt.subplots(ncols=2)  
  
ax[0].imshow(x_train[0])  
ax[0].set_title('Before Preprocessing')  
ax[1].imshow(tf.squeeze(pre_processing_v1(x_train[:1, :, :])))  
ax[1].set_title('After Preprocessing')  
plt.show()
```

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).



```
In [ ]: x_train = x_train.astype('float32')  
x_train /= (255/2)  
x_train -= 1
```

Building Models

In a generative adversarial network [GAN] there will 2 networks in play. A generator and a discriminator.

Generator: Produce synthetic data samples similar to the training data
Discriminator: Distinguish samples from real dataset to the generator

Both networks plays a game of MinMax where the generator wants to minimize the difference image to make it look real and the discriminator wants to maximize the ability to spot the real from the fakes.

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

We will be building a few deep learning models to generate and identify real or fake images.

Architecture List:

1. DCGAN (Baseline)
2. cGAN
3. ACGAN

However before we create the generator, we should look through the different models that can be used to make our discriminator too.

As the discriminator is only predicting real image vs fake image, the discriminator will be a CNN model.

GAN Model Evaluation Methods

There are several methods to evaluate the GAN. However, there are some issues with evaluating a GAN generator model.

1. There is no labels for GAN, the main focus is the quality of the synthetic data.
2. Different models have different objective functions

There are a few methods:

1. Manual GAN Evaluation

- The simplest evaluation method is through visual evaluation using our eyes.
- However, there are some issues:
 - A. This method is very subjective and have some bias by the reviewer
 - B. As the Cifar-10 images are only (32, 32, 3), it is hard to differentiate the classes and it is very small which is unclear
 - C. If there is a large quantity of images generated, there is no way to go through each and every image to evaluate it

2. Qualitative GAN Evaluation

- By identifying non numerical data in the images, we are able to use human subject evaluation or evaluation via comparison. Some techniques include using:
 - A. Nearest Neighbors [Detect overfitting, generated samples are shown next to nearest neighbour in the training set]
 - B. Rapid Scene Categorization [Get participants to help classify real and fake data]

3. Quantitative GAN Evaluation

- Evaluating GAN on quantitative means refers to using a specific calculation of specific numerical scores, used to summarize the general quality of the images.
- There are a few metrics:
 - A. KL Divergence (aka relative entropy) [Measures the distance between two statistical probability]
 - B. Fréchet Inception Distance [compares distribution of generated images]

For simplicity, we will be only evaluating the GAN models with manual and quantitative evaluation

Fréchet inception distance (FID)

Fréchet inception distance (FID) is a metric that calculates the distance between feature vectors calculated for real and generated images. Basically it shows how closely related the generated image and real image are to one another. The lower the score the more correlated the images are [Might suggest higher quality images].

$$FID = \|\mu - \mu_w\|_2^2 + \text{tr}(C + C_w - 2(CC_w)^{\frac{1}{2}})$$

We also need to take note that FID score may not be the most accurate as FID score requires a large dataset to be very precise as FID uses covariance to calculate the score. Further more FID score is biased as the InceptionV3 model used to extract features are corrupted by color, intensity, saturation, and horizontal translations. Another issue with the FID score is we are unable to tell which classes are generated better.

However, as FID is one of the most common GAN evaluation metrics, we will ignore this issues with FID score and continue using this as our comparing metrics

```
In [ ]: class GAN_FID:
    def __init__(self, batch_size, noise, sample_size, buffer_size, labels=False):
        # setting Hyperparameters
        self.BATCH_SIZE = batch_size
        self.noise = noise
        self.SAMPLE_SIZE = sample_size
        self.BUFFER_SIZE = buffer_size
        self.labels = labels

        # setting Constants
        self.INCEPTION_SHAPE = (299, 299, 3)
        self.INCEPTION = InceptionV3(
            include_top=False, pooling='avg', input_shape=self.INCEPTION_SHAPE)
        self.AUTO = tf.data.AUTOTUNE

    # method to set generator and training data
    def fit(self, generator, train_data):
        # setting generative model and original data used for training
        self.GENERATOR = generator
        self.train_data = train_data

        # Preparing Real Images
        trainloader = tf.data.Dataset.from_tensor_slices((self.train_data))
        trainloader = (
            trainloader
            .shuffle(self.BUFFER_SIZE)
            .map(self.__resize_and_preprocess, num_parallel_calls=self.AUTO)
            .batch(self.BATCH_SIZE, num_parallel_calls=self.AUTO)
            .prefetch(self.AUTO)
        )
        self.trainloader = trainloader

        if self.labels:
            # Generate and prepare Synthetic Images (Fake)
            rand_labels = np.random.randint(
                low=0, high=10, size=self.SAMPLE_SIZE)
            rand_labels = to_categorical(rand_labels)
            noise = tf.random.normal([self.SAMPLE_SIZE, self.noise])
            if self.labels:
                generated_images = self.GENERATOR([noise, rand_labels])
            else:
                generated_images = self.GENERATOR(noise)
            genloader = tf.data.Dataset.from_tensor_slices(generated_images)
            genloader = (
                genloader
                .map(self.__resize_and_preprocess, num_parallel_calls=self.AUTO)
                .batch(self.BATCH_SIZE, num_parallel_calls=self.AUTO)
                .prefetch(self.AUTO)
            )
            self.genloader = genloader

            # prepare embeddings
            count = math.ceil(self.SAMPLE_SIZE/self.BATCH_SIZE)

            # compute embeddings for real images
            print("Computing Real Image Embeddings")
            self.real_image_embeddings = self.__compute_embeddings(
                self.trainloader, count)

            # compute embeddings for generated images
            print("Computing Generated Image Embeddings")
```

```

        self.generated_image_embeddings = self.__compute_embeddings(
            self.genloader, count)
    assert self.real_image_embeddings.shape == self.generated_image_embeddings.shape, "Em
    print("Computed Embeddings\tReal Images Embedding Shape: {} \tGenerated Images Embeddi
        self.real_image_embeddings.shape,
        self.generated_image_embeddings.shape
    )))

# method to produce evaluation results
@tf.autograph.experimental.do_not_convert
def evaluate(self):
    # calculate Frechet Inception Distance
    fid = self.__calculate_fid(
        self.real_image_embeddings, self.generated_image_embeddings)
    print('The computed FID score is:', fid)

    return fid

# method to generate embeddings from inception model
def __compute_embeddings(self, dataloader, count):
    image_embeddings = []
    for _ in tqdm(range(count)):
        images = next(iter(dataloader))
        embeddings = self.INCEPTION.predict(images)
        image_embeddings.extend(embeddings)
    return np.array(image_embeddings)

# STATIC METHODS: these methods knows nothing about the class
# static method to prepare the data before computing Inception Embeddings
@staticmethod
def __resize_and_preprocess(image):
    # image *= 255.0 # original image are scaled to [0, 1], scaling back to [0, 255]
    image -= -1
    image /= (1 - (-1))
    image *= 255.

    # .preprocess_input() expects an image of scale [0, 255]
    image = preprocess_input(image)
    # inception model expects an image of shape (None, 299, 299, 3)
    image = tf.image.resize(image, (299, 299), method='nearest')
    return image

# static method to calculate frechet inception distance based on embeddings
@staticmethod
def __calculate_fid(real_embeddings, generated_embeddings):
    # calculate mean and covariance statistics
    mu1, sigma1 = real_embeddings.mean(
        axis=0), np.cov(real_embeddings, rowvar=False)
    mu2, sigma2 = generated_embeddings.mean(
        axis=0), np.cov(generated_embeddings, rowvar=False)
    # calculate sum squared difference between means
    ssdiff = np.sum((mu1 - mu2)**2.0)
    # calculate sqrt of product between cov
    covmean = sqrtm(sigma1.dot(sigma2))
    # check and correct imaginary numbers from sqrt
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    # calculate score
    fid = ssdiff + np.trace(sigma1 + sigma2 - 2.0 * covmean)
    return fid

```

In []: # Hyperparameters and Constants
BATCH_SIZE = 64
AUTO = tf.data.AUTOTUNE
EPOCHS = 100

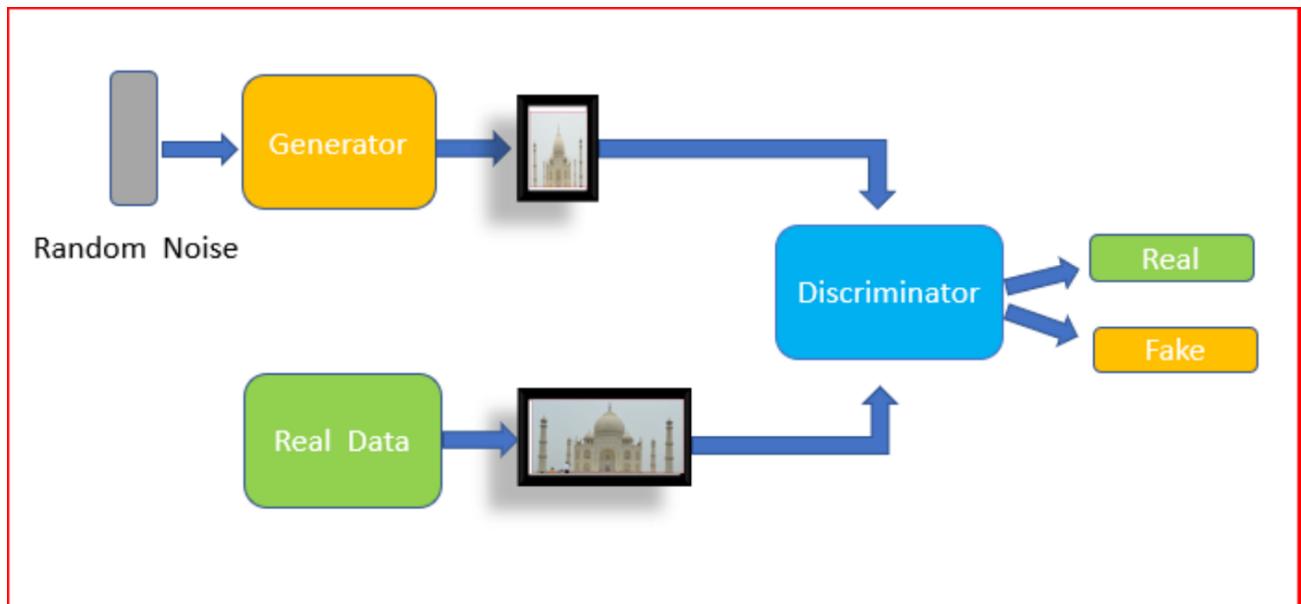
```
BUFFER_SIZE = 1024  
NOISE = 128
```

DCGAN (Baseline)

For a GAN architecture, it consist of 2 different models.

1. Discriminator
2. Generator

Both Discriminator and Generator will play a min max game. The generator wants to minimize and ensure that the image generated looks real. The discriminator wants to maximize and ensure that it can distinguish samples from real and fake and wants to spot the fake.



DCGAN Discriminator

Basically, the discriminator is a binary classifier. For the DCGAN model, we will make a typical CNN network. However, unlike a typical CNN network, we will be modifying it such that DCGAN model will be improved.

1. Pooling Layers in a classic CNN are used to down sample activation features maps. However, this is not recommended for GAN as information will be loss. Therefore, we will be using a 2 x 2 Stride Convolution
2. Due to the vanishing gradient problem, the activation function for a GAN is usually a LeakyReLU. It allows gradients to flow easier through the architecture. This allows the function to compute the greatest value between the features and a small factor.
3. As sparse gradients will occur and diminish over time [random spikes in the loss during training], there will be many numerical operations performed on them. To fix this, we will add BatchNormalization to the network so that it can reduce covariate shifts. BatchNormalization will be used before non-Gaussian activations like LeakyReLU.
4. One issue of implementing GAN is the unstable training issue. The issue happens when the discriminator becomes more powerful and overwhelming the generator. We will normalise the weights and controls the Lipschitz constant of the discriminator to mitigate the exploding gradient problem and the mode collapse problem. [Spectral Normalization]
5. He and Xavier weights initialization are unsuitable for training as it makes training more unstable due to the larger distribution of weights. To reduce the instability of training, we will be using Gaussian Weight Initialization[mean = 0, std = 0.02]

```
In [ ]: # function to create discriminator model
def create_discriminator(image_size):
    weights_init = RandomNormal(mean=0, stddev=0.02)
    discriminator = Sequential([
        Input(shape=IMG_SIZE),
        SpectralNormalization(
            Conv2D(64, kernel_size=4, strides=2, padding='same',
                   kernel_initializer=weights_init)
        ),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        SpectralNormalization(
            Conv2D(128, kernel_size=4, strides=2, padding='same',
                   kernel_initializer=weights_init)
        ),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        SpectralNormalization(
            Conv2D(256, kernel_size=4, strides=2, padding='same',
                   kernel_initializer=weights_init)
        ),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        SpectralNormalization(
            Conv2D(512, kernel_size=4, strides=2, padding='same',
                   kernel_initializer=weights_init)
        ),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        GlobalMaxPooling2D(),
        Dense(1, activation='sigmoid'),
    ], name='discriminator_GAN')
    return discriminator

create_discriminator(image_size=IMG_SIZE).summary()
```

Model: "discriminator_GAN"

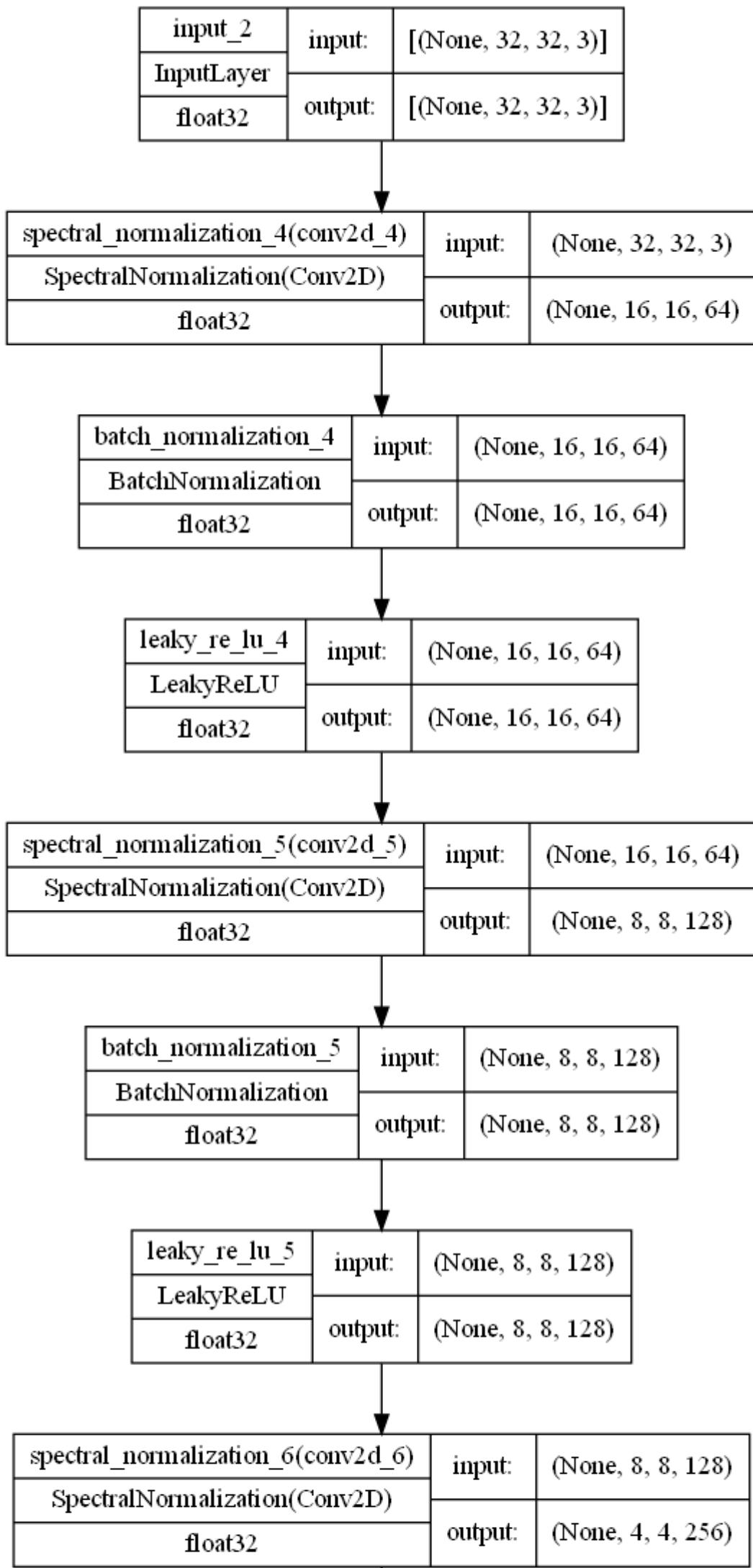
Layer (type)	Output Shape	Param #
=====		
spectral_normalization_12 (SpectralNormalization)	(None, 16, 16, 64)	3200
batch_normalization_154 (BatchNormalization)	(None, 16, 16, 64)	256
leaky_re_lu_44 (LeakyReLU)	(None, 16, 16, 64)	0
spectral_normalization_13 (SpectralNormalization)	(None, 8, 8, 128)	131328
batch_normalization_155 (BatchNormalization)	(None, 8, 8, 128)	512
leaky_re_lu_45 (LeakyReLU)	(None, 8, 8, 128)	0
spectral_normalization_14 (SpectralNormalization)	(None, 4, 4, 256)	524800
batch_normalization_156 (BatchNormalization)	(None, 4, 4, 256)	1024
leaky_re_lu_46 (LeakyReLU)	(None, 4, 4, 256)	0
spectral_normalization_15 (SpectralNormalization)	(None, 2, 2, 512)	2098176
batch_normalization_157 (BatchNormalization)	(None, 2, 2, 512)	2048
leaky_re_lu_47 (LeakyReLU)	(None, 2, 2, 512)	0
global_max_pooling2d (GlobalMaxPooling2D)	(None, 512)	0
dense_15 (Dense)	(None, 1)	513
=====		
Total params:	2,761,857	
Trainable params:	2,758,977	
Non-trainable params:	2,880	

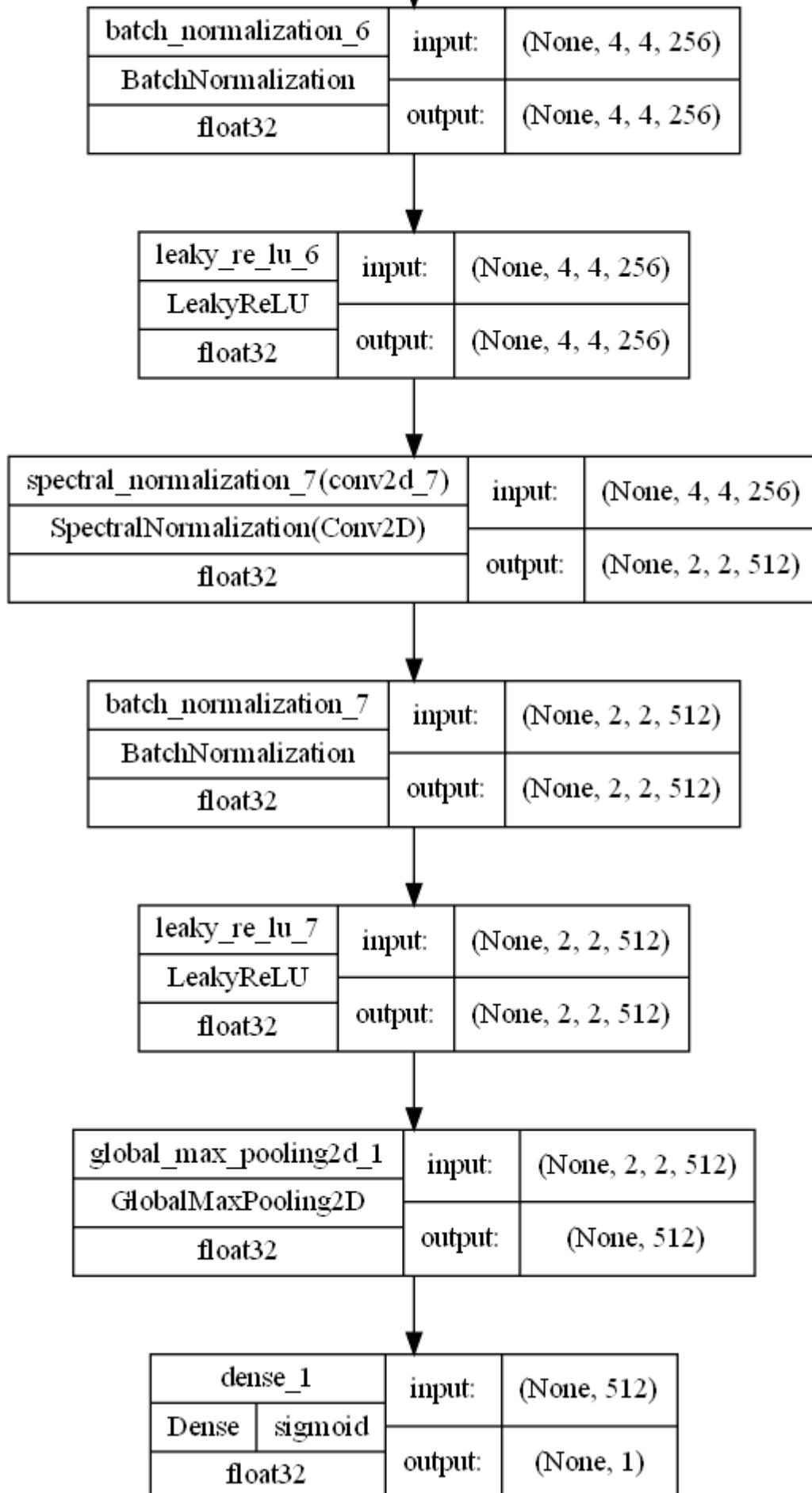
```
c:\Users\Soh Hong Yu\anaconda3\envs\gpu_env\lib\site-packages\keras\initializers\initializers_v2.py:120: UserWarning: The initializer RandomNormal is unseeded and being called multiple times, which will return identical values each time (even if the initializer is unseeded). Please update your code to provide a seed to the initializer, or avoid using the same initializer instance more than once.
```

```
    warnings.warn(
```

```
In [ ]: plot_model(create_discriminator(image_size=IMG_SIZE), to_file="images/models/DCGAN_discrimina  
show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True
```

Out[]:





DCGAN Generator

Unlike your typical neural architectures, the generator will have multiple convolutional upscaled.

1. Transposed Convolution Layers allows the normal convolution to go in the opposite direction. The main purpose is to allow the CNN to upsample from the noise latent vector to generated the images.

2. As sparse gradients will occur and diminish over time [random spikes in the loss during training], there will be many numerical operations performed on them. To fix this, we will add BatchNormalization to the network so that it can reduce covariate shifts.
3. He and Xavier weights initialization are unsuitable for training as it makes training more unstable due to the larger distribution of weights. To reduce the instability of training, we will be using Gaussian Weight Initialization[mean = 0, std = 0.02]
4. As we are using the LeakyReLU in the discriminator, and typically we will use ReLU in the generator. However, due to the sparse gradient during training and effects like the vanishing gradient problems, we can mitigate this effects using PReLU which as introduced with learning parameter to help training.

```
In [ ]: def create_generator(noise):

    # gaussian weights initialization
    weights_init = RandomNormal(mean=0, stddev=0.02)

    generator = Sequential([
        # fc block: handling Latent vector z
        Input(shape=(noise,)),
        Dense(2*2*512),
        Reshape((2, 2, 512)),

        # Conv 1
        # using a kernel size that is a factor of the stride
        Conv2DTranspose(256, kernel_size=4, strides=2,
                        padding='same', kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        PReLU(),

        # Conv 2
        Conv2DTranspose(128, kernel_size=4, strides=2,
                        padding='same', kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        PReLU(),

        # Conv 3
        Conv2DTranspose(64, kernel_size=4, strides=2,
                        padding='same', kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        PReLU(),

        # Output Layer
        Conv2DTranspose(3, kernel_size=4, strides=2, padding='same',
                        activation='tanh', kernel_initializer=weights_init)
    ], name='generator_GAN')
    return generator

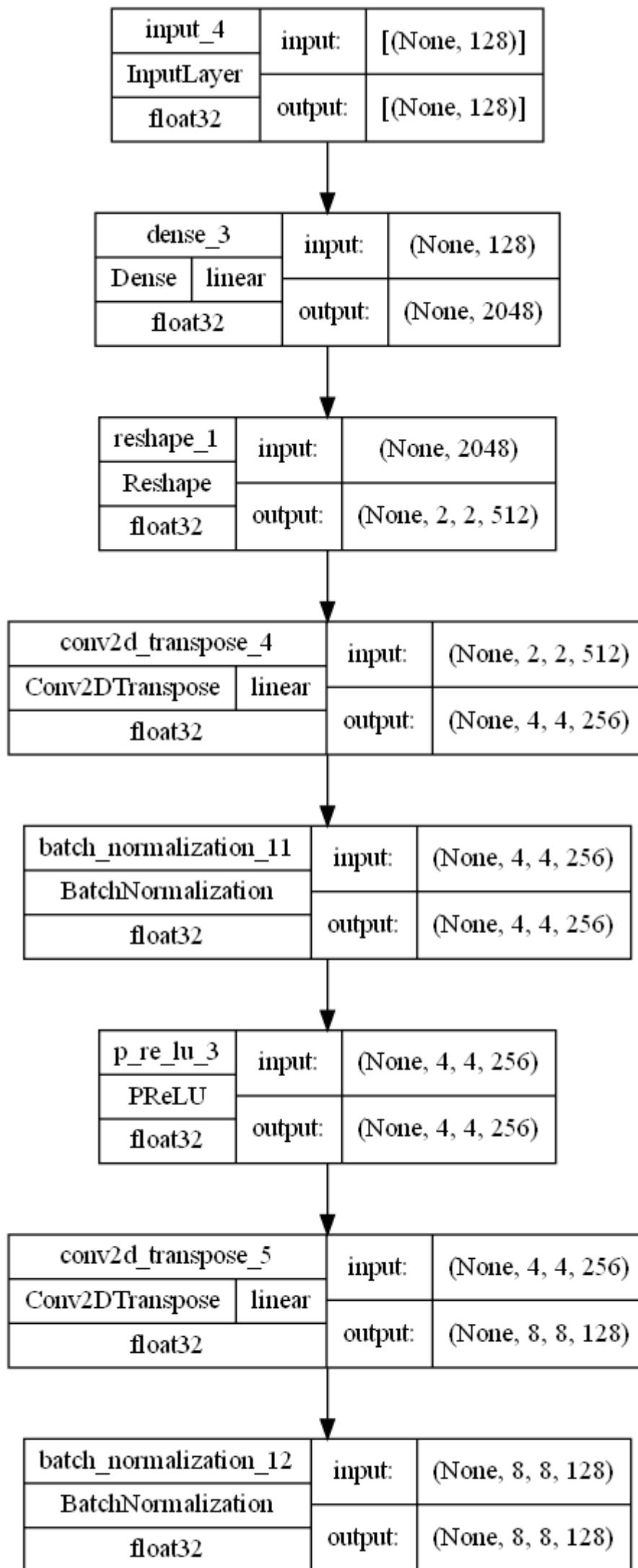
create_generator(noise=NOISE).summary()
```

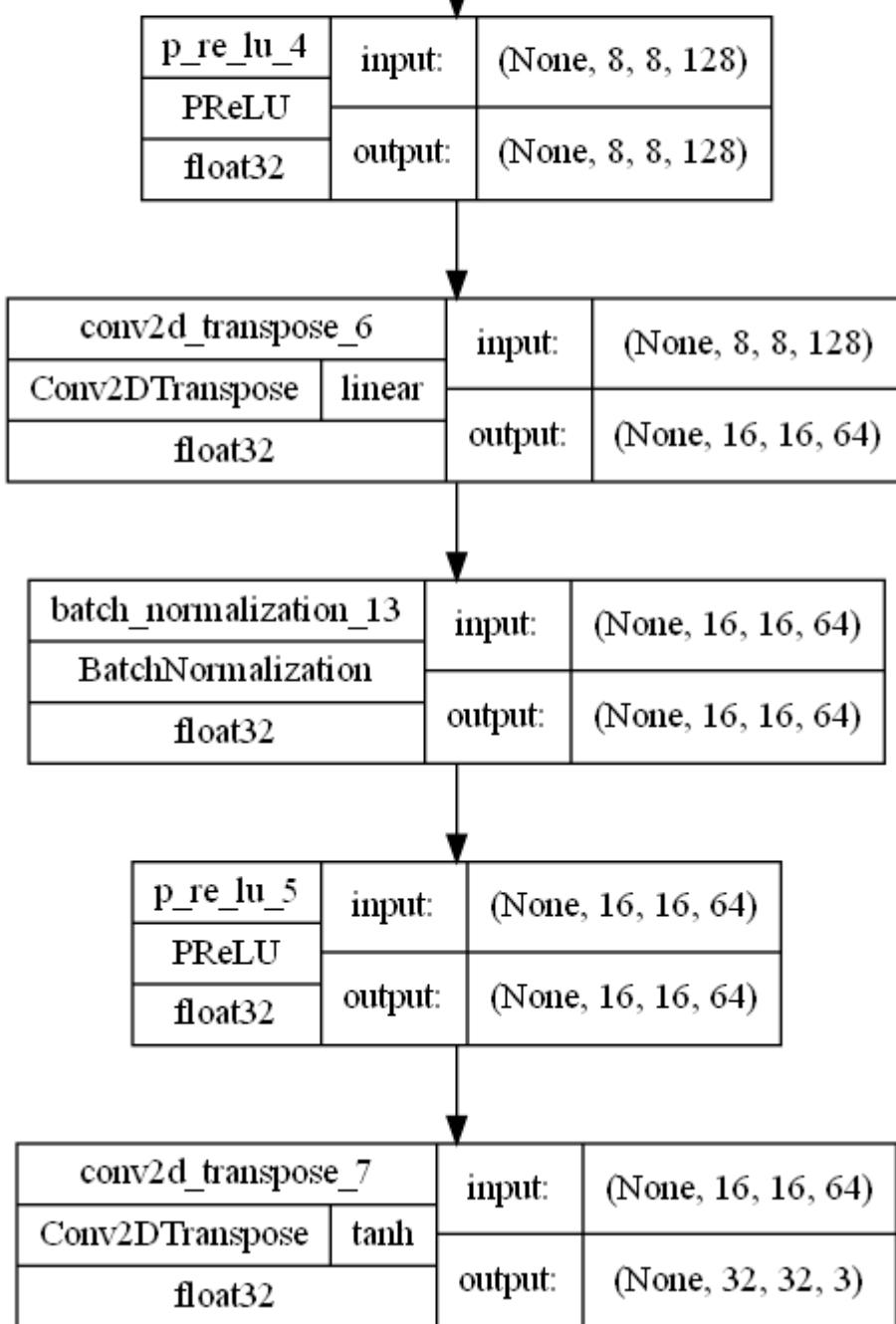
Model: "generator_GAN"

Layer (type)	Output Shape	Param #
=====		
dense_16 (Dense)	(None, 2048)	264192
reshape_6 (Reshape)	(None, 2, 2, 512)	0
conv2d_transpose_18 (Conv2D Transpose)	(None, 4, 4, 256)	2097408
=====		
dense_16 (Dense)	(None, 2048)	264192
reshape_6 (Reshape)	(None, 2, 2, 512)	0
conv2d_transpose_18 (Conv2D Transpose)	(None, 4, 4, 256)	2097408
batch_normalization_158 (BatchNormalization)	(None, 4, 4, 256)	1024
p_re_lu_16 (PReLU)	(None, 4, 4, 256)	4096
conv2d_transpose_19 (Conv2D Transpose)	(None, 8, 8, 128)	524416
batch_normalization_159 (BatchNormalization)	(None, 8, 8, 128)	512
p_re_lu_17 (PReLU)	(None, 8, 8, 128)	8192
conv2d_transpose_20 (Conv2D Transpose)	(None, 16, 16, 64)	131136
batch_normalization_160 (BatchNormalization)	(None, 16, 16, 64)	256
p_re_lu_18 (PReLU)	(None, 16, 16, 64)	16384
conv2d_transpose_21 (Conv2D Transpose)	(None, 32, 32, 3)	3075
=====		
Total params:	3,050,691	
Trainable params:	3,049,795	
Non-trainable params:	896	

```
In [ ]: plot_model(create_generator(noise=NOISE), to_file="images/models/DCGAN_generator.png", show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True)
```

Out[]:

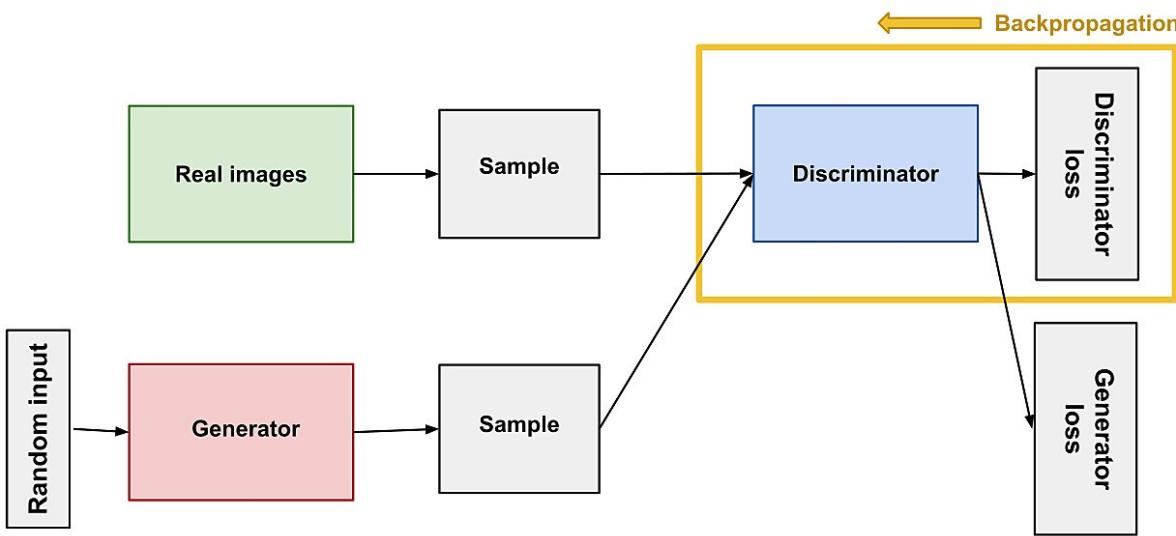




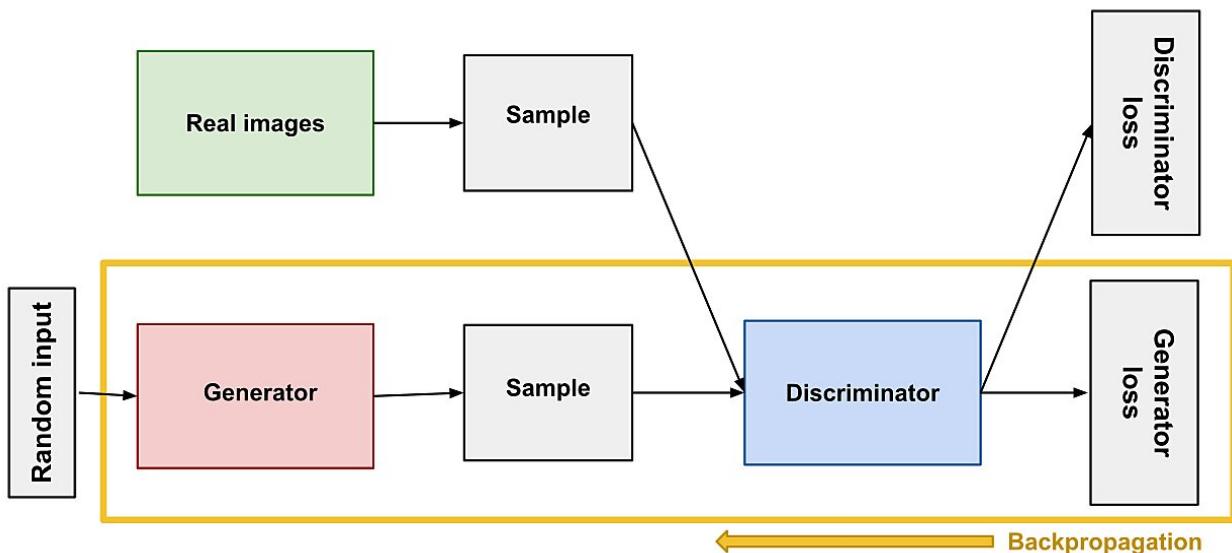
GAN Training Process

To train the GAN model, we will be making a class. As the training process is quite repetitive, we will be going to describe the process in a single iteration.

1. First the Generator will generate a batch of samples using a noise vector
2. Next the Discriminator will classify the real images and generated images [1 and 0].
3. Subsequently, the Discriminator will update the weights



4. The Generator will generate another batch of samples using the noise vector
5. Discriminator will compute a discrimination using the samples
6. The loss function is then applied between the discrimination computed previously
7. The Generator's weights will then be updated



To create the GAN Model, we will be referencing the TensorFlow website to help wrap a GAN model.

https://www.tensorflow.org/guide/keras/customizing_what_happens_in_fit#wrapping_up_an_end-to-end_gan_example

```
In [ ]: class GAN(Model):
    def __init__(self, discriminator, generator, noise):
        super(GAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.noise = noise
        self.gen_loss_tracker = keras.metrics.Mean(name='generator_loss')
        self.disc_loss_tracker = keras.metrics.Mean(name='discriminator_loss')
        self.kl = tf.keras.metrics.KLDivergence()

    def compile(self, d_optimizer, g_optimizer, loss_fn):
        super(GAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.loss_fn = loss_fn
```

```

def train_step(self, real_images):
    if isinstance(real_images, tuple):
        real_images = real_images[0]
    # Sample random points in the latent space
    batch_size = tf.shape(real_images)[0]
    random_latent_vectors = tf.random.normal(
        shape=(batch_size, self.noise))
    # Decode them to fake images
    generated_images = self.generator(random_latent_vectors)
    # Combine them with real images
    combined_images = tf.concat([generated_images, real_images], axis=0)

    # Assemble Labels discriminating real from fake images
    labels = tf.concat(
        [tf.zeros((batch_size, 1)), tf.ones((batch_size, 1))], axis=0
    )
    # Add random noise to the labels - important trick!
    labels += 0.05 * tf.random.uniform(tf.shape(labels))

    # Train the discriminator
    with tf.GradientTape() as tape:
        predictions = self.discriminator(combined_images)
        d_loss = self.loss_fn(labels, predictions)
    grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
    self.d_optimizer.apply_gradients(
        zip(grads, self.discriminator.trainable_weights)
    )

    # TRAINING GENERATOR
    # Sample random points in the latent space
    random_latent_vectors = tf.random.normal(
        shape=(batch_size, self.noise))

    # Assemble Labels that say "all real images"
    misleading_labels = tf.ones((batch_size, 1))

    # Train the generator (note that we should *not* update the weights
    # of the discriminator)!
    with tf.GradientTape() as tape:
        generated_images = self.generator(random_latent_vectors)
        predictions = self.discriminator(generated_images)
        g_loss = self.loss_fn(misleading_labels, predictions)
    grads = tape.gradient(g_loss, self.generator.trainable_weights)
    self.g_optimizer.apply_gradients(
        zip(grads, self.generator.trainable_weights))

    # Monitor Loss
    self.disc_loss_tracker.update_state(d_loss)
    self.gen_loss_tracker.update_state(g_loss)
    self.kl.update_state(y_true=real_images, y_pred=generated_images)

    return {
        "d_loss": self.disc_loss_tracker.result(),
        "g_loss": self.gen_loss_tracker.result(),
        "KL Divergence": self.kl.result(),
    }

```

DCGAN Callback

To help monitor the DCGAN's loss and progress and visualise the images after every patience. We will need to make a custom tensorflow callback that will help monitor the DCGAN.

```
In [ ]: class GANMonitor(keras.callbacks.Callback):
    def __init__(self, num_img=10, noise=128, patience=10, vmin=0, vmax=1):
        self.num_img = num_img
```

```

        self.noise = noise
        self.patience = patience
        self.vmin = vmin
        self.vmax = vmax
        self.constant_noise = tf.random.normal(
            shape=(self.num_img, self.noise))

    def generate_plot(self):
        # Generate Images
        generated_images = self.model.generator(self.constant_noise)
        # Normalise Image from [vmin, vmax] to [0, 1]
        generated_images -= self.vmin
        generated_images /= (self.vmax - self.vmin)
        row_size = int(np.ceil(self.num_img/5))
        fig = plt.figure(figsize=(10, 2*row_size), tight_layout=True)
        for i in range(self.num_img):
            ax = fig.add_subplot(row_size, 5, i+1)
            ax.imshow(generated_images[i])
            ax.axis('off')
        plt.show()

    def save_weights(self, epoch=None):
        try:
            if epoch != None:
                name = 'GAN/generator-epoch-{}.h5'.format(epoch)
                print('Generator Checkpoint - {}'.format(name))
                self.model.generator.save_weights(
                    filepath=name,
                    save_format='h5'
                )
        except Exception as e:
            print(e)

    def on_epoch_begin(self, epoch, logs=None):
        if epoch % self.patience == 0:
            self.generate_plot()
            self.save_weights(epoch)

    def on_train_end(self, epoch, logs=None):
        self.generate_plot()
        self.save_weights('Full Train')

```

```
In [ ]: callbacks = [
    GANMonitor(num_img=15, noise=128, patience=5, vmin=-1, vmax=1),
]
```

DCGAN Training

After setting the GAN architecture, we can now run the GAN models to generate images. First, we need to convert the data into a tensorflow dataset and training using multiprocessing to speed up process.

```

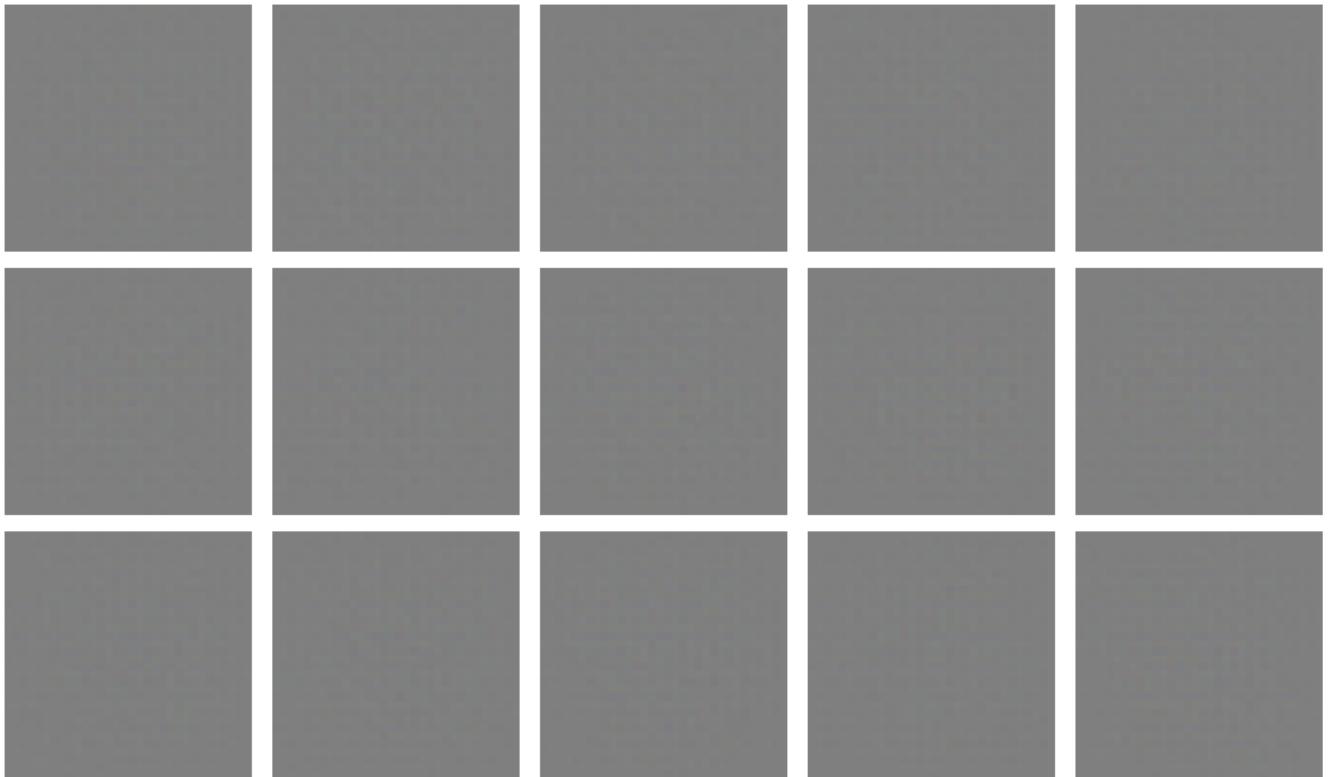
In [ ]: tf.keras.backend.clear_session()
K.clear_session()

gan = GAN(
    discriminator=create_discriminator(image_size=IMG_SIZE),
    generator=create_generator(noise=NOISE),
    noise=NOISE
)

gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=2e-4, beta_1=0.5),
    g_optimizer=keras.optimizers.Adam(learning_rate=2e-4, beta_1=0.5),
    loss_fn=keras.losses.BinaryCrossentropy(),

```

```
)  
  
dataset = tf.data.Dataset.from_tensor_slices(x_train)  
dataset = dataset.shuffle(buffer_size=BUFFER_SIZE).batch(  
    batch_size=BATCH_SIZE, num_parallel_calls=AUTO).prefetch(AUTO)  
  
gan_hist = gan.fit(dataset, epochs=EPOCHS, use_multiprocessing=True,  
                    workers=16, callbacks=callbacks)
```



```
Generator Checkpoint - GAN/generator-epoch-0.h5  
Epoch 1/100  
782/782 [=====] - 95s 116ms/step - d_loss: 0.8099 - g_loss: 2.3560 -  
KL Divergence: 5.5728  
Epoch 2/100  
782/782 [=====] - 91s 116ms/step - d_loss: 0.5243 - g_loss: 1.3519 -  
KL Divergence: 5.0051  
Epoch 3/100  
782/782 [=====] - 90s 115ms/step - d_loss: 0.5362 - g_loss: 1.3995 -  
KL Divergence: 4.9535  
Epoch 4/100  
782/782 [=====] - 78s 99ms/step - d_loss: 0.4980 - g_loss: 1.6549 -  
KL Divergence: 5.0642  
Epoch 5/100  
782/782 [=====] - 77s 98ms/step - d_loss: 0.4545 - g_loss: 1.7139 -  
KL Divergence: 4.9792
```



Generator Checkpoint - GAN/generator-epoch-5.h5

Epoch 6/100

782/782 [=====] - 75s 96ms/step - d_loss: 0.4542 - g_loss: 1.7561 -
KL Divergence: 5.2424

Epoch 7/100

782/782 [=====] - 72s 92ms/step - d_loss: 0.4260 - g_loss: 1.8319 -
KL Divergence: 5.3081

Epoch 8/100

782/782 [=====] - 77s 99ms/step - d_loss: 0.3847 - g_loss: 1.9351 -
KL Divergence: 5.3142

Epoch 9/100

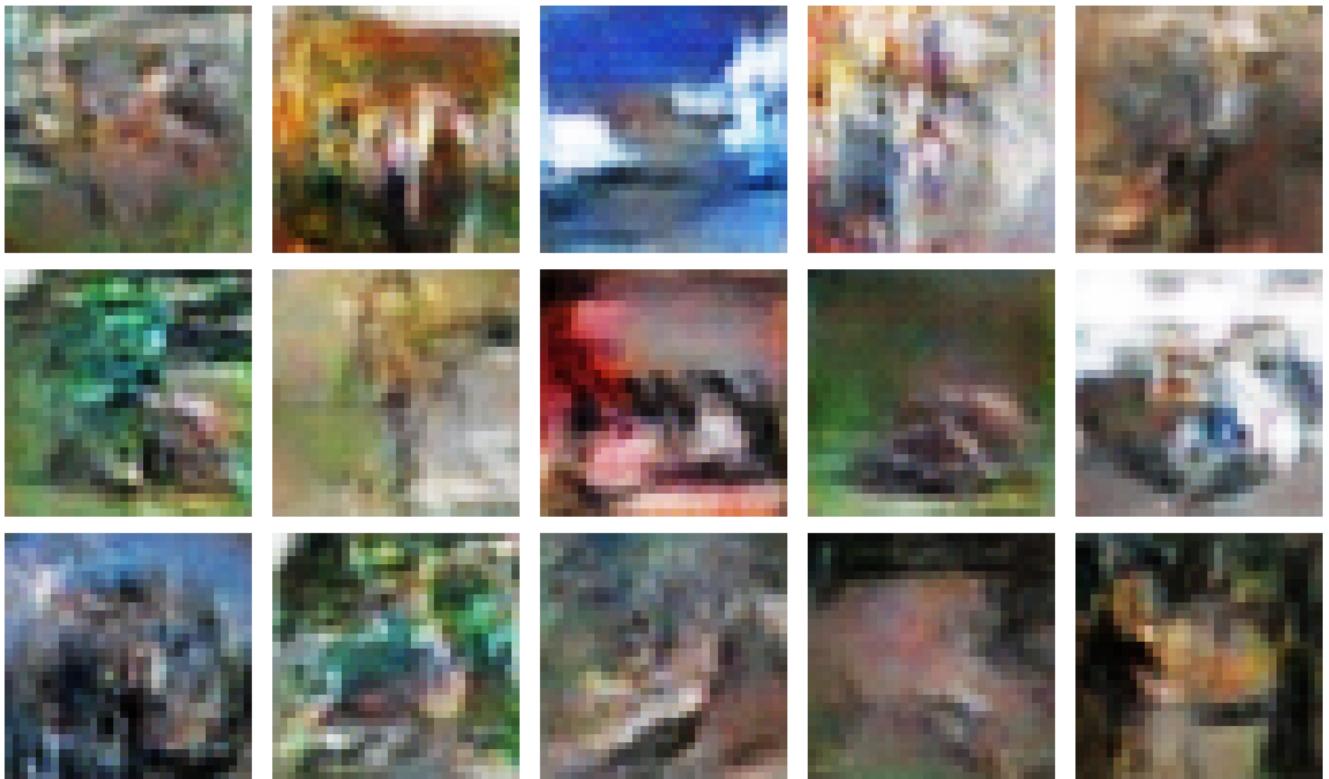
782/782 [=====] - 80s 103ms/step - d_loss: 0.4222 - g_loss: 1.7614 -
KL Divergence: 5.2157

Epoch 10/100

782/782 [=====] - 74s 94ms/step - d_loss: 0.4308 - g_loss: 1.6405 -
KL Divergence: 5.2183



Generator Checkpoint - GAN/generator-epoch-10.h5
Epoch 11/100
782/782 [=====] - 72s 93ms/step - d_loss: 0.4341 - g_loss: 1.5846 -
KL Divergence: 5.2427
Epoch 12/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.4353 - g_loss: 1.5529 -
KL Divergence: 5.2889
Epoch 13/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.4247 - g_loss: 1.6162 -
KL Divergence: 5.2508
Epoch 14/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.4312 - g_loss: 1.5784 -
KL Divergence: 5.2568
Epoch 15/100
782/782 [=====] - 65s 84ms/step - d_loss: 0.4375 - g_loss: 1.5366 -
KL Divergence: 5.2752



Generator Checkpoint - GAN/generator-epoch-15.h5
Epoch 16/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.4261 - g_loss: 1.5810 -
KL Divergence: 5.3126
Epoch 17/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.4012 - g_loss: 1.6312 -
KL Divergence: 5.3076
Epoch 18/100
782/782 [=====] - 66s 84ms/step - d_loss: 0.3830 - g_loss: 1.7145 -
KL Divergence: 5.3971
Epoch 19/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.3504 - g_loss: 1.8088 -
KL Divergence: 5.4369
Epoch 20/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.3303 - g_loss: 1.9310 -
KL Divergence: 5.5025



Generator Checkpoint - GAN/generator-epoch-20.h5

Epoch 21/100

782/782 [=====] - 65s 83ms/step - d_loss: 0.3051 - g_loss: 2.0590 -
KL Divergence: 5.5511

Epoch 22/100

782/782 [=====] - 65s 83ms/step - d_loss: 0.2950 - g_loss: 2.1144 -
KL Divergence: 5.5393

Epoch 23/100

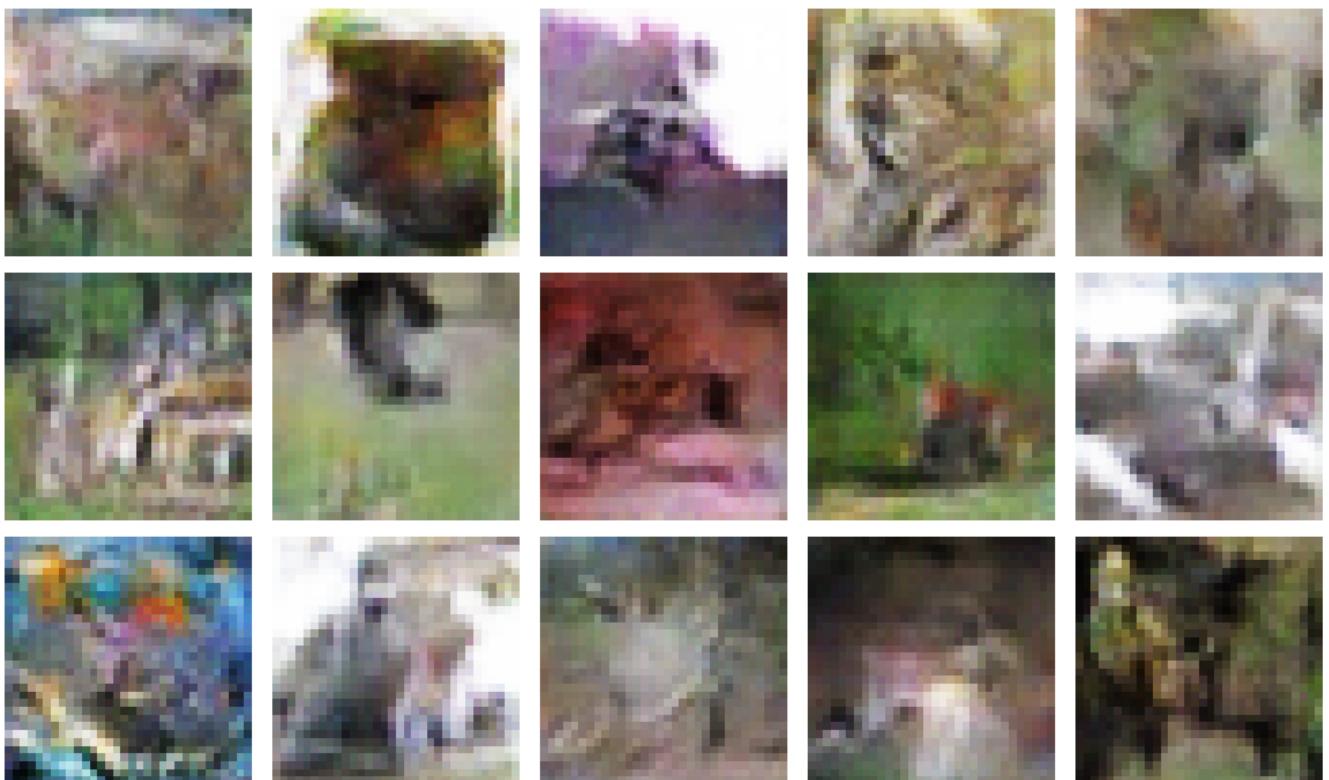
782/782 [=====] - 65s 83ms/step - d_loss: 0.3249 - g_loss: 2.0975 -
KL Divergence: 5.4939

Epoch 24/100

782/782 [=====] - 65s 84ms/step - d_loss: 0.2348 - g_loss: 2.2961 -
KL Divergence: 5.5796

Epoch 25/100

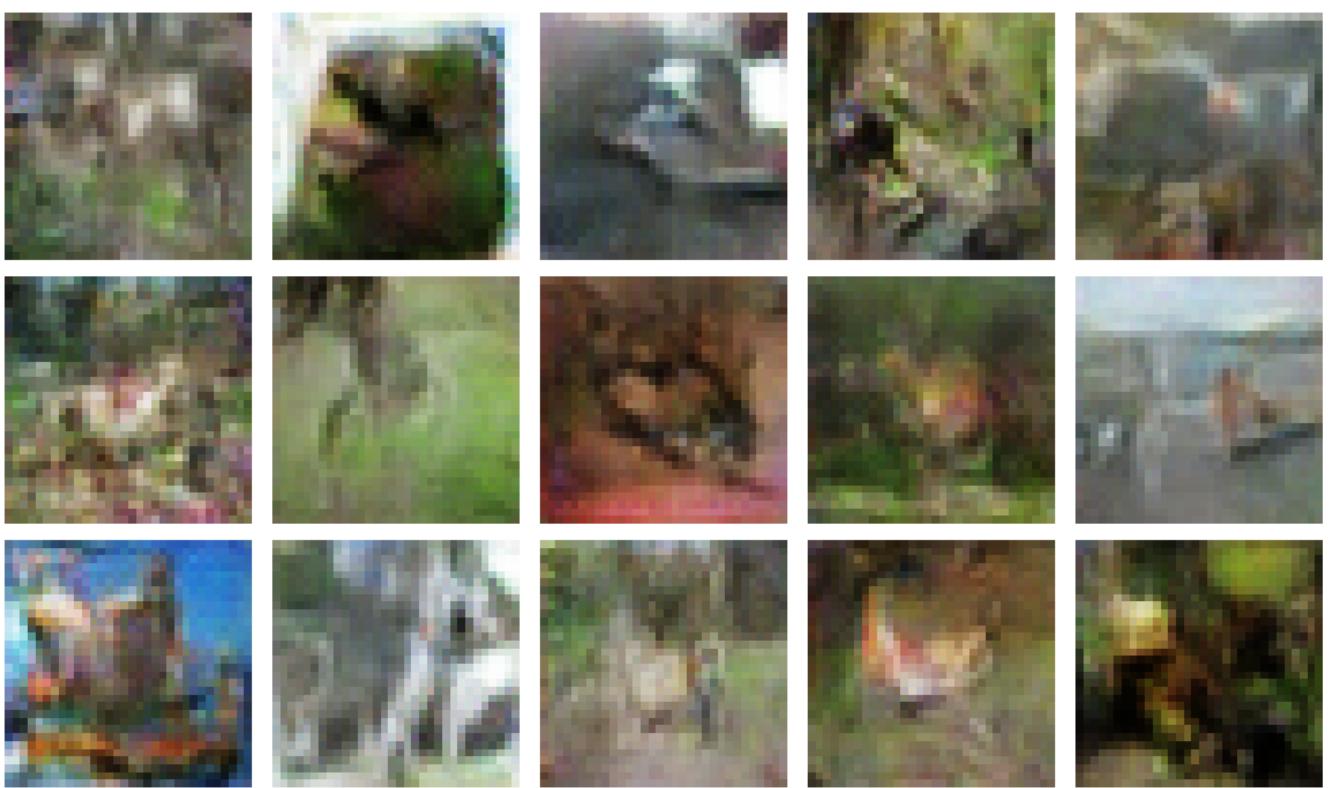
782/782 [=====] - 65s 84ms/step - d_loss: 0.2958 - g_loss: 2.2357 -
KL Divergence: 5.5292



Generator Checkpoint - GAN/generator-epoch-25.h5
Epoch 26/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.2135 - g_loss: 2.4048 -
KL Divergence: 5.4618
Epoch 27/100
782/782 [=====] - 65s 84ms/step - d_loss: 0.2243 - g_loss: 2.3866 -
KL Divergence: 5.5467
Epoch 28/100
782/782 [=====] - 65s 84ms/step - d_loss: 0.2290 - g_loss: 2.4840 -
KL Divergence: 5.5828
Epoch 29/100
782/782 [=====] - 65s 84ms/step - d_loss: 0.1924 - g_loss: 2.5711 -
KL Divergence: 5.6314
Epoch 30/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.2399 - g_loss: 2.5162 -
KL Divergence: 5.5612



Generator Checkpoint - GAN/generator-epoch-30.h5
Epoch 31/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.2327 - g_loss: 2.5737 -
KL Divergence: 5.4968
Epoch 32/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.1736 - g_loss: 2.7707 -
KL Divergence: 5.5980
Epoch 33/100
782/782 [=====] - 65s 84ms/step - d_loss: 0.1370 - g_loss: 2.8270 -
KL Divergence: 5.5297
Epoch 34/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.1860 - g_loss: 2.8384 -
KL Divergence: 5.5760
Epoch 35/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.2088 - g_loss: 2.8062 -
KL Divergence: 5.4036



Generator Checkpoint - GAN/generator-epoch-35.h5

Epoch 36/100

782/782 [=====] - 65s 83ms/step - d_loss: 0.1440 - g_loss: 2.9751 -
KL Divergence: 5.6307

Epoch 37/100

782/782 [=====] - 65s 83ms/step - d_loss: 0.1361 - g_loss: 2.9243 -
KL Divergence: 5.4859

Epoch 38/100

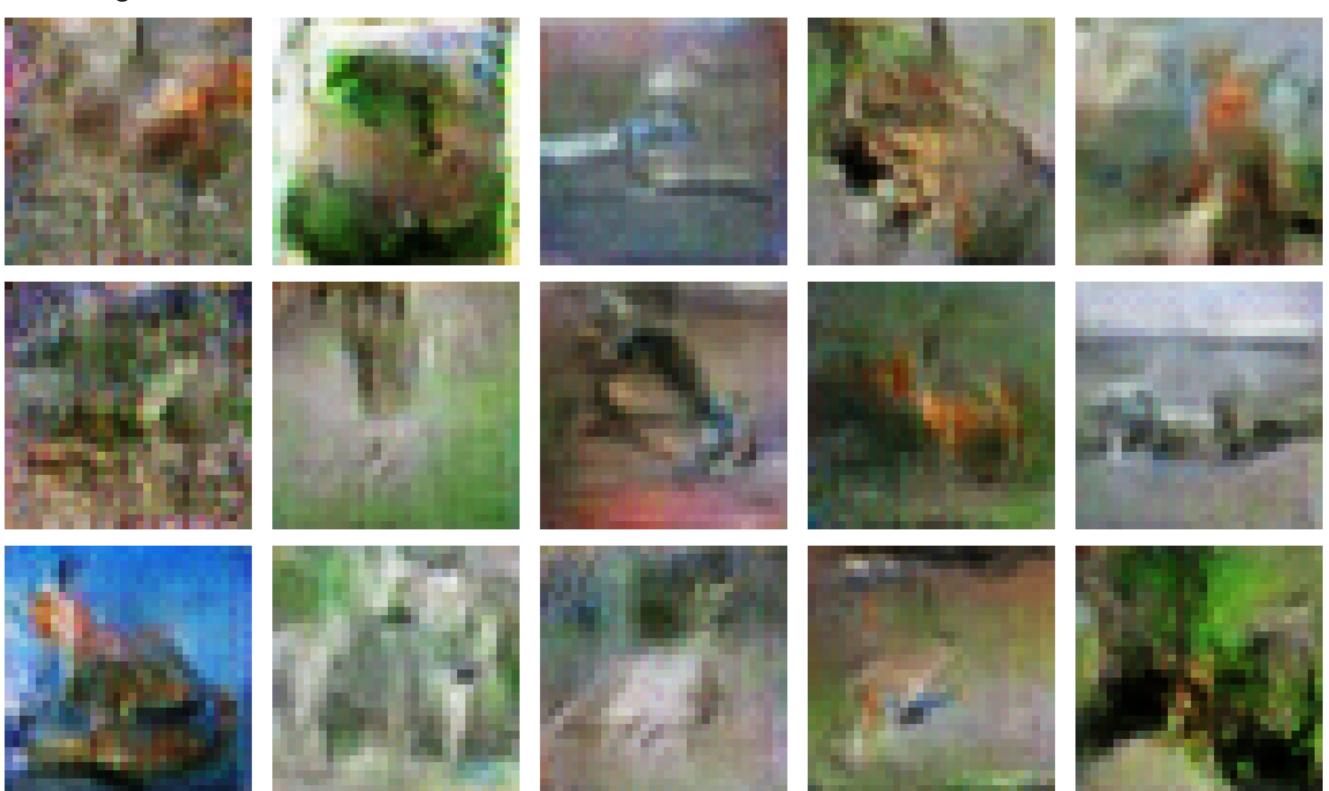
782/782 [=====] - 65s 83ms/step - d_loss: -0.0034 - g_loss: 2.9355 -
KL Divergence: 5.6491

Epoch 39/100

782/782 [=====] - 65s 83ms/step - d_loss: -0.1112 - g_loss: 3.0740 -
KL Divergence: 5.4985

Epoch 40/100

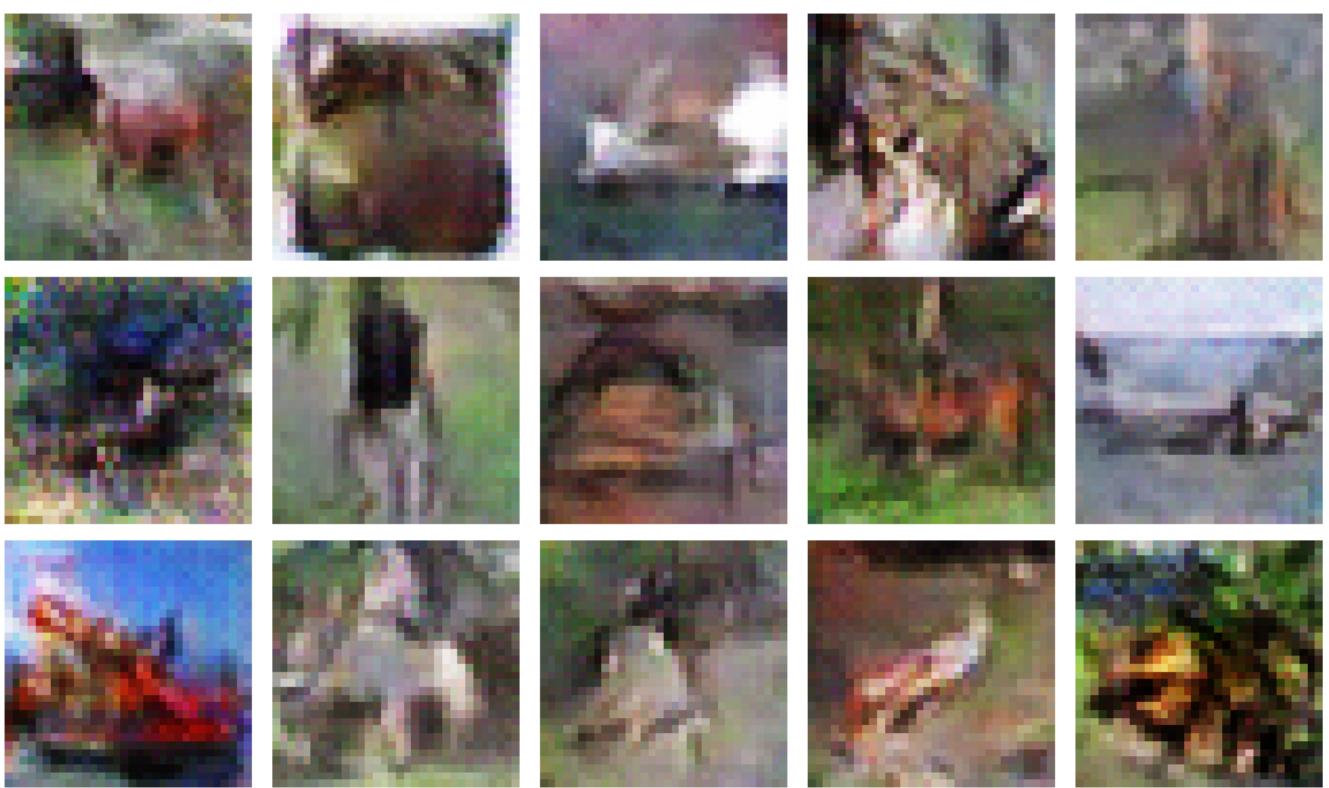
782/782 [=====] - 65s 83ms/step - d_loss: -0.1806 - g_loss: 3.0779 -
KL Divergence: 5.4900



Generator Checkpoint - GAN/generator-epoch-40.h5
Epoch 41/100
782/782 [=====] - 65s 83ms/step - d_loss: -0.2689 - g_loss: 2.9071 -
KL Divergence: 5.3409
Epoch 42/100
782/782 [=====] - 65s 83ms/step - d_loss: -0.6264 - g_loss: 2.7827 -
KL Divergence: 5.4605
Epoch 43/100
782/782 [=====] - 65s 84ms/step - d_loss: -0.7570 - g_loss: 2.6962 -
KL Divergence: 5.4708
Epoch 44/100
782/782 [=====] - 65s 84ms/step - d_loss: 0.0150 - g_loss: 2.8504 -
KL Divergence: 5.3204
Epoch 45/100
782/782 [=====] - 65s 83ms/step - d_loss: -0.3417 - g_loss: 2.6018 -
KL Divergence: 5.2870



Generator Checkpoint - GAN/generator-epoch-45.h5
Epoch 46/100
782/782 [=====] - 65s 83ms/step - d_loss: -1.5169 - g_loss: 2.7410 -
KL Divergence: 5.4491
Epoch 47/100
782/782 [=====] - 65s 84ms/step - d_loss: -1.1741 - g_loss: 2.4387 -
KL Divergence: 5.5688
Epoch 48/100
782/782 [=====] - 65s 83ms/step - d_loss: -0.7234 - g_loss: 2.8247 -
KL Divergence: 5.3680
Epoch 49/100
782/782 [=====] - 65s 84ms/step - d_loss: -3.7880 - g_loss: 3.0092 -
KL Divergence: 5.3943
Epoch 50/100
782/782 [=====] - 65s 83ms/step - d_loss: -1.7728 - g_loss: 2.7732 -
KL Divergence: 5.2406



Generator Checkpoint - GAN/generator-epoch-50.h5

Epoch 51/100

782/782 [=====] - 65s 83ms/step - d_loss: -1.4409 - g_loss: 2.9775 -
KL Divergence: 5.2834

Epoch 52/100

782/782 [=====] - 65s 83ms/step - d_loss: -7.8287 - g_loss: 3.3714 -
KL Divergence: 5.3293

Epoch 53/100

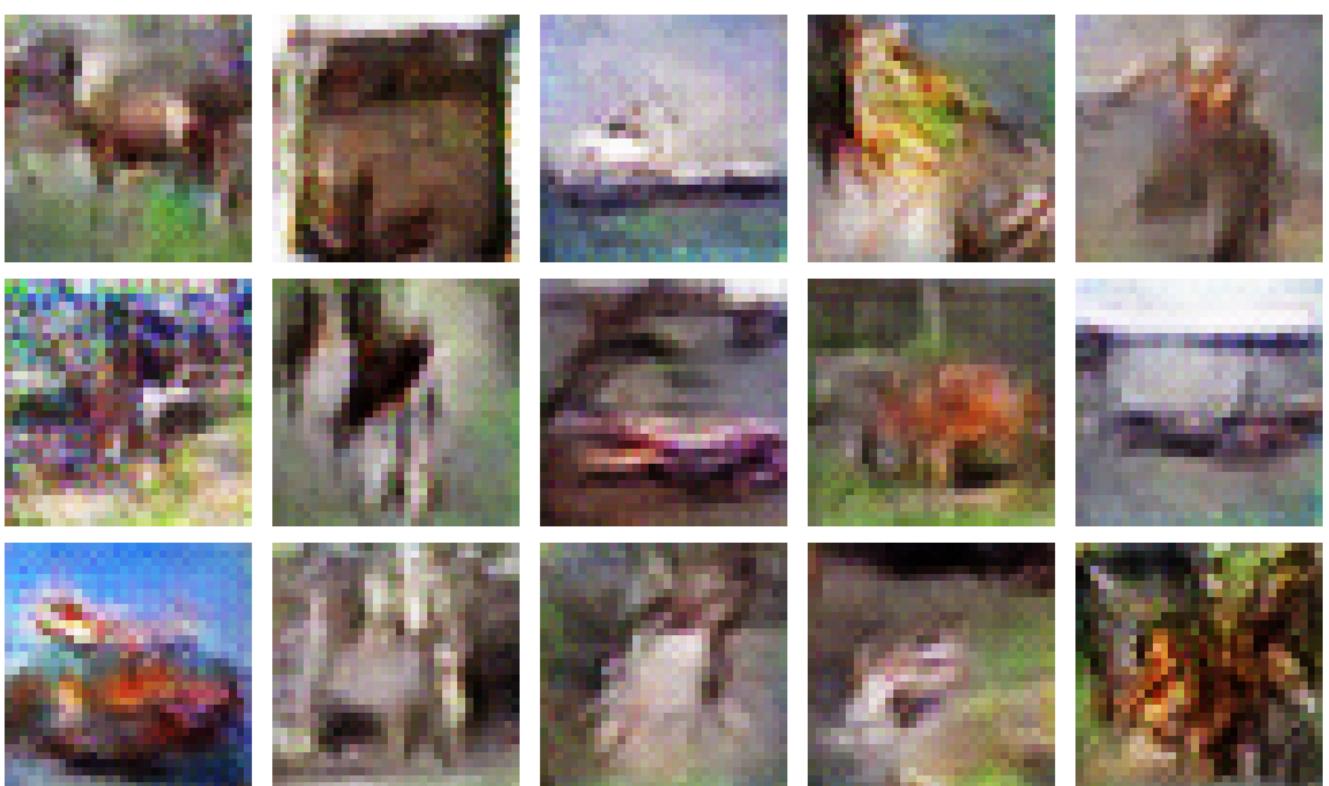
782/782 [=====] - 65s 83ms/step - d_loss: -7.6940 - g_loss: 3.7294 -
KL Divergence: 5.2441

Epoch 54/100

782/782 [=====] - 65s 83ms/step - d_loss: -5.6276 - g_loss: 3.3253 -
KL Divergence: 5.1040

Epoch 55/100

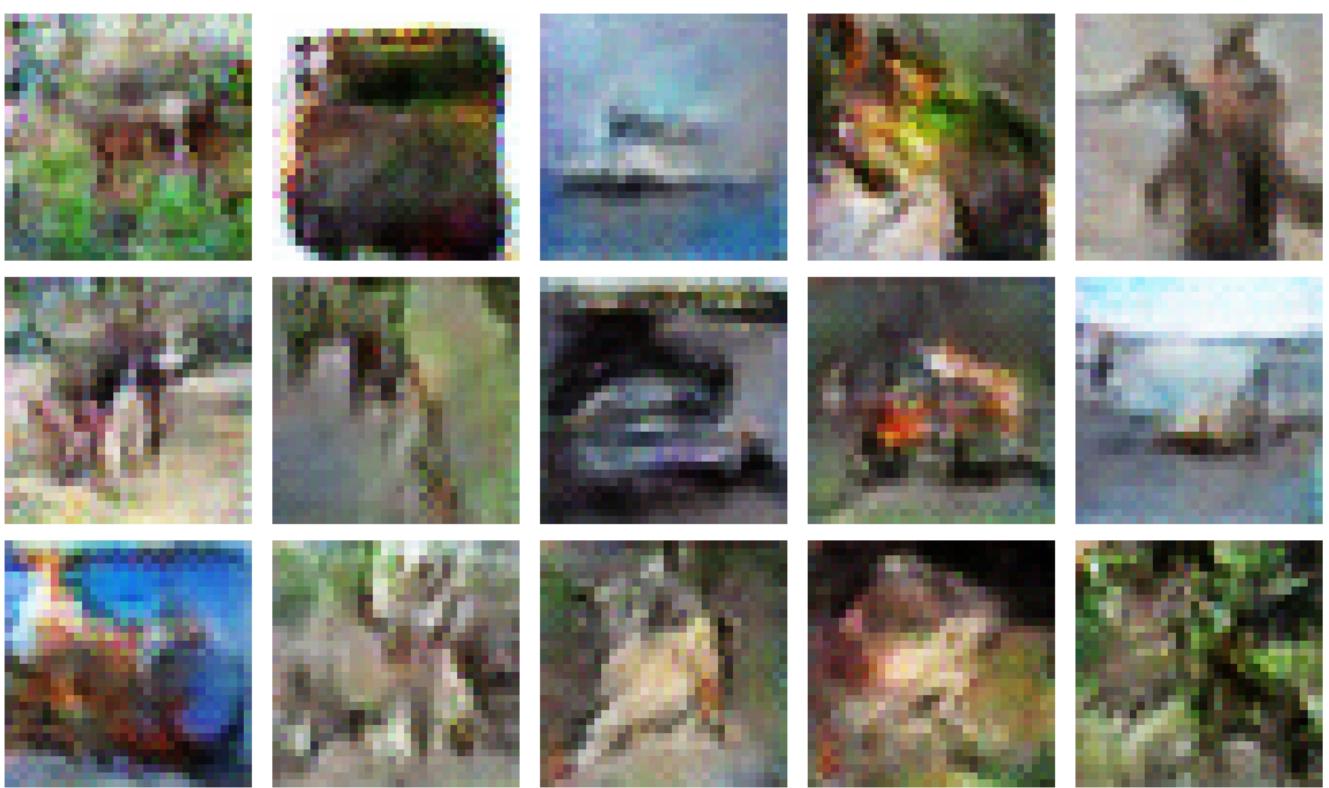
782/782 [=====] - 65s 83ms/step - d_loss: -6.5953 - g_loss: 4.3503 -
KL Divergence: 5.0761



Generator Checkpoint - GAN/generator-epoch-55.h5
Epoch 56/100
782/782 [=====] - 65s 83ms/step - d_loss: 3.8748 - g_loss: 4.5775 -
KL Divergence: 5.1706
Epoch 57/100
782/782 [=====] - 65s 83ms/step - d_loss: -1.5376 - g_loss: 3.7667 -
KL Divergence: 5.2501
Epoch 58/100
782/782 [=====] - 65s 83ms/step - d_loss: -9.7839 - g_loss: 4.6667 -
KL Divergence: 5.1637
Epoch 59/100
782/782 [=====] - 65s 83ms/step - d_loss: -17.7642 - g_loss: 6.0499 -
- KL Divergence: 5.1694
Epoch 60/100
782/782 [=====] - 65s 82ms/step - d_loss: -1.4228 - g_loss: 6.8363 -
KL Divergence: 5.0922



Generator Checkpoint - GAN/generator-epoch-60.h5
Epoch 61/100
782/782 [=====] - 65s 83ms/step - d_loss: -21.4890 - g_loss: 5.6337 -
- KL Divergence: 5.1796
Epoch 62/100
782/782 [=====] - 65s 83ms/step - d_loss: 0.4711 - g_loss: 6.6734 -
KL Divergence: 5.1199
Epoch 63/100
782/782 [=====] - 65s 84ms/step - d_loss: -2.3045 - g_loss: 6.2231 -
KL Divergence: 5.0101
Epoch 64/100
782/782 [=====] - 65s 83ms/step - d_loss: -2.6852 - g_loss: 6.2505 -
KL Divergence: 4.9846
Epoch 65/100
782/782 [=====] - 65s 83ms/step - d_loss: -8.7825 - g_loss: 7.3908 -
KL Divergence: 5.1555



Generator Checkpoint - GAN/generator-epoch-65.h5

Epoch 66/100

782/782 [=====] - 65s 83ms/step - d_loss: 1.6436 - g_loss: 7.2768 -
KL Divergence: 5.1522

Epoch 67/100

782/782 [=====] - 65s 83ms/step - d_loss: -21.2175 - g_loss: 7.4385
- KL Divergence: 5.1556

Epoch 68/100

782/782 [=====] - 65s 83ms/step - d_loss: -56.5155 - g_loss: 9.7812
- KL Divergence: 5.1574

Epoch 69/100

782/782 [=====] - 65s 83ms/step - d_loss: -24.4689 - g_loss: 9.5014
- KL Divergence: 5.1998

Epoch 70/100

782/782 [=====] - 65s 83ms/step - d_loss: -68.9150 - g_loss: 11.8571
- KL Divergence: 5.0571



Generator Checkpoint - GAN/generator-epoch-70.h5
Epoch 71/100
782/782 [=====] - 65s 83ms/step - d_loss: -40.4674 - g_loss: 12.1600
- KL Divergence: 5.0973
Epoch 72/100
782/782 [=====] - 65s 83ms/step - d_loss: -54.0762 - g_loss: 15.8519
- KL Divergence: 5.0757
Epoch 73/100
782/782 [=====] - 65s 83ms/step - d_loss: -102.1723 - g_loss: 13.686
6 - KL Divergence: 5.0720
Epoch 74/100
782/782 [=====] - 65s 83ms/step - d_loss: -3.3449 - g_loss: 20.1484
- KL Divergence: 5.1252
Epoch 75/100
782/782 [=====] - 65s 83ms/step - d_loss: -30.8211 - g_loss: 16.9026
- KL Divergence: 5.0135



Generator Checkpoint - GAN/generator-epoch-75.h5
Epoch 76/100
782/782 [=====] - 65s 83ms/step - d_loss: -134.2528 - g_loss: 18.639
7 - KL Divergence: 5.0382
Epoch 77/100
782/782 [=====] - 65s 83ms/step - d_loss: -25.5029 - g_loss: 23.1385
- KL Divergence: 5.0396
Epoch 78/100
782/782 [=====] - 65s 83ms/step - d_loss: 8.7221 - g_loss: 22.2647 -
KL Divergence: 5.3234
Epoch 79/100
782/782 [=====] - 65s 83ms/step - d_loss: -27.8002 - g_loss: 21.4541
- KL Divergence: 5.1387
Epoch 80/100
782/782 [=====] - 65s 83ms/step - d_loss: -39.1104 - g_loss: 22.9841
- KL Divergence: 5.0516



Generator Checkpoint - GAN/generator-epoch-80.h5

Epoch 81/100

782/782 [=====] - 65s 84ms/step - d_loss: -65.5465 - g_loss: 26.5511
- KL Divergence: 5.1373

Epoch 82/100

782/782 [=====] - 65s 84ms/step - d_loss: -45.9117 - g_loss: 24.1248
- KL Divergence: 5.1034

Epoch 83/100

782/782 [=====] - 65s 84ms/step - d_loss: -145.6371 - g_loss: 26.217
7 - KL Divergence: 5.1188

Epoch 84/100

782/782 [=====] - 65s 83ms/step - d_loss: -40.8575 - g_loss: 31.3308
- KL Divergence: 5.0919

Epoch 85/100

782/782 [=====] - 65s 83ms/step - d_loss: -150.8906 - g_loss: 29.578
2 - KL Divergence: 5.0584



Generator Checkpoint - GAN/generator-epoch-85.h5
Epoch 86/100
782/782 [=====] - 65s 83ms/step - d_loss: -208.3862 - g_loss: 42.635
2 - KL Divergence: 5.2663
Epoch 87/100
782/782 [=====] - 65s 83ms/step - d_loss: -123.9887 - g_loss: 37.449
5 - KL Divergence: 5.0143
Epoch 88/100
782/782 [=====] - 65s 83ms/step - d_loss: -38.3129 - g_loss: 46.1979
- KL Divergence: 4.9254
Epoch 89/100
782/782 [=====] - 64s 82ms/step - d_loss: -30.6947 - g_loss: 37.9809
- KL Divergence: 5.1031
Epoch 90/100
782/782 [=====] - 65s 83ms/step - d_loss: -133.7166 - g_loss: 36.007
9 - KL Divergence: 5.0970



Generator Checkpoint - GAN/generator-epoch-90.h5
Epoch 91/100
782/782 [=====] - 65s 83ms/step - d_loss: -164.8235 - g_loss: 42.481
1 - KL Divergence: 5.0938
Epoch 92/100
782/782 [=====] - 65s 83ms/step - d_loss: -173.5644 - g_loss: 46.255
8 - KL Divergence: 5.0844
Epoch 93/100
782/782 [=====] - 65s 83ms/step - d_loss: -243.3874 - g_loss: 52.177
1 - KL Divergence: 5.0237
Epoch 94/100
782/782 [=====] - 65s 83ms/step - d_loss: -8.8408 - g_loss: 60.3328
- KL Divergence: 5.0516
Epoch 95/100
782/782 [=====] - 65s 83ms/step - d_loss: -203.5086 - g_loss: 60.046
7 - KL Divergence: 5.1207



Generator Checkpoint - GAN/generator-epoch-95.h5

Epoch 96/100

782/782 [=====] - 65s 83ms/step - d_loss: 80.9257 - g_loss: 81.5817
- KL Divergence: 4.8600

Epoch 97/100

782/782 [=====] - 65s 83ms/step - d_loss: -47.3651 - g_loss: 53.4179
- KL Divergence: 4.9050

Epoch 98/100

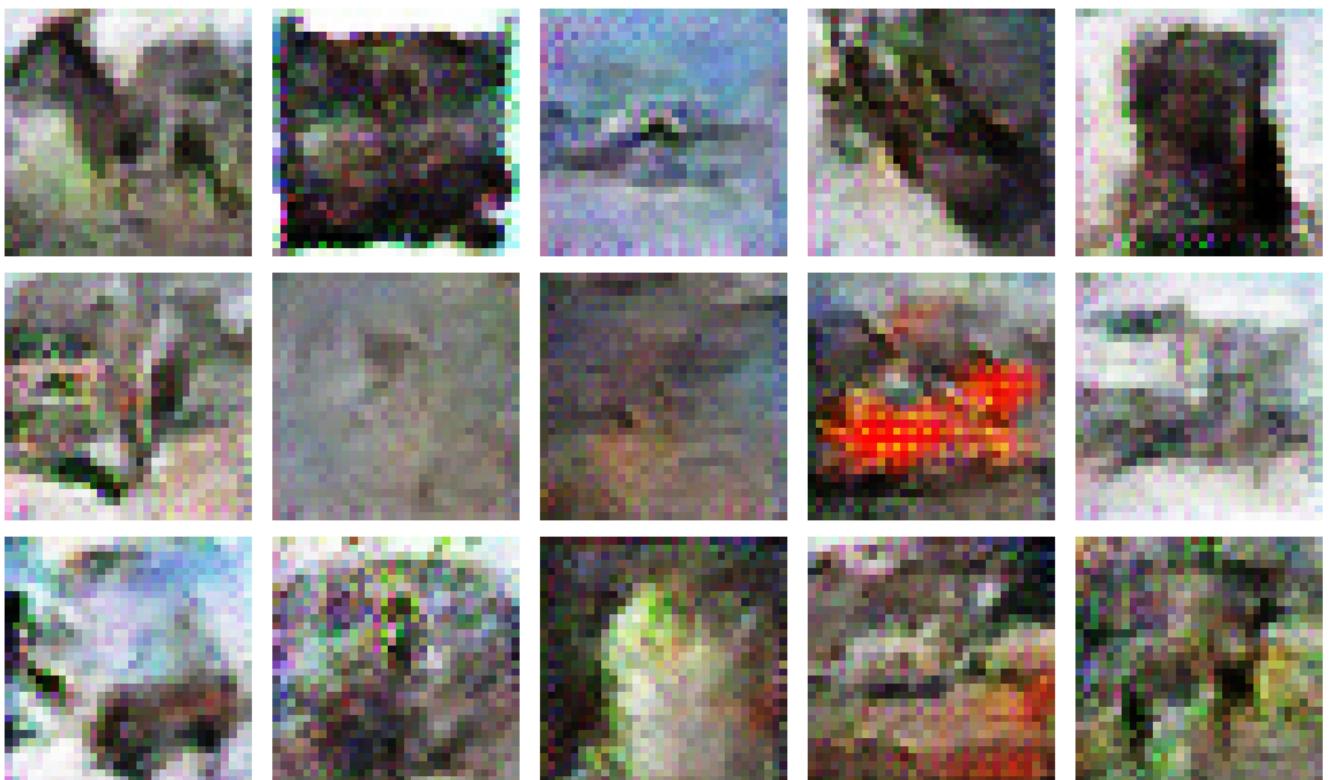
782/782 [=====] - 65s 83ms/step - d_loss: -51.3024 - g_loss: 51.2221
- KL Divergence: 4.9625

Epoch 99/100

782/782 [=====] - 65s 83ms/step - d_loss: -260.3817 - g_loss: 56.398
5 - KL Divergence: 5.0082

Epoch 100/100

782/782 [=====] - 65s 83ms/step - d_loss: -75.4194 - g_loss: 59.0767
- KL Divergence: 5.1662



Generator Checkpoint - GAN/generator-epoch-Full Train.h5

Observations

We note that the images that generated at around 45 epochs are very clear with no clear signs of noise. But afterwards, we can see that noise starts showing up.

A few reasons why noise starts appearing in GAN includes:

1. Model collapse: When the generator produces a limited set of outputs, the model is unable to capture the diversity of the training data. This results in the generator producing samples that are variations of a few distinct modes or patterns, and the noise appearing in the generated images.
2. High variance in the data: GANs can struggle to generate high-quality images when there is a high variance in the training data. This can cause the generator to produce images with a lot of noise or artifacts.
3. Poorly chosen hyperparameters: GANs require careful tuning of hyperparameters such as learning rate, batch size, and network architecture. If the hyperparameters are not chosen correctly, the generator may produce images with a lot of noise.
4. Overfitting: GANs, like any other neural networks, can overfit to the training data, meaning that it will memorize the training examples and not generalize well to new examples. This can lead to the generator producing images with a lot of noise.
5. Unstable training: GANs can be difficult to train due to their instability during the training process. This can lead to the generator producing images with a lot of noise.

For this case, we can see that the model issue is likely due to the model collapsing.

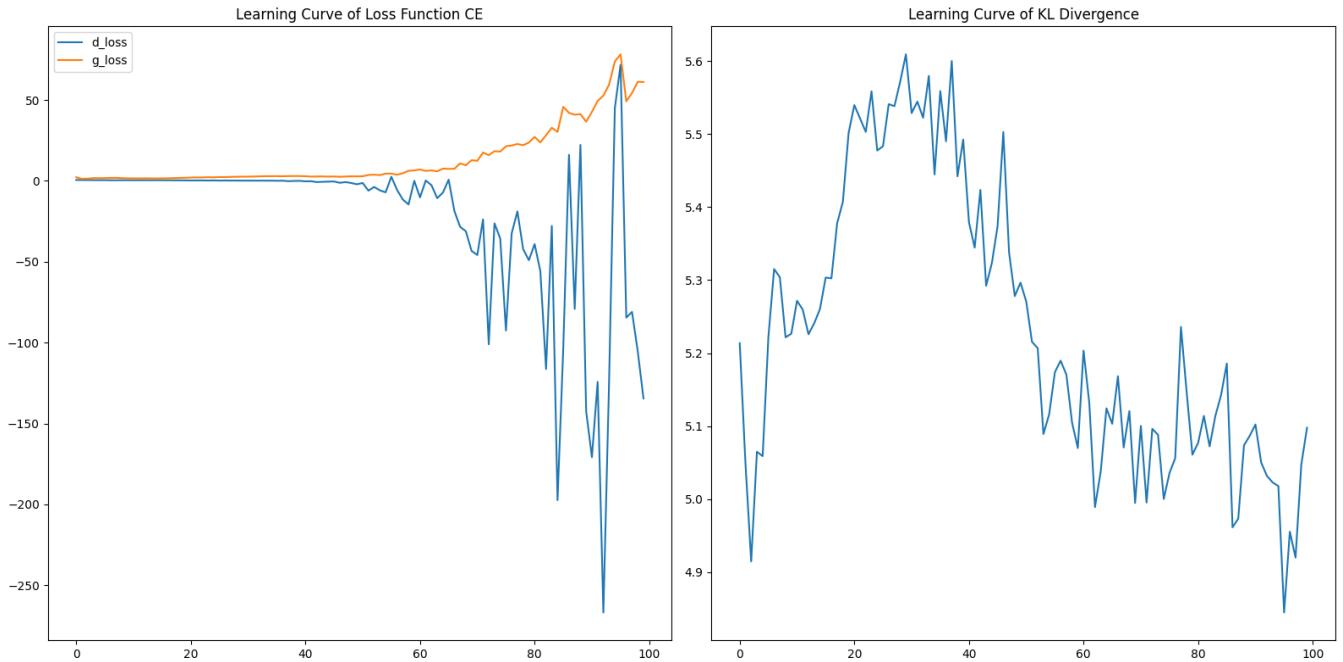
DCGAN Evaluation

As mentioned previously we will be using a quantitative and manual evaluation of the DCGAN.

We will start off with a quantitative analysis so that we can find the best model to generate the images for manual evaluation.

```
In [ ]: # store history object into dataframe
gan_hist_df = pd.DataFrame(gan_hist.history)

# using pandas dataframe to plot out Learning curve
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8), tight_layout=True)
gan_hist_df.loc[:, ["d_loss", 'g_loss']].plot(
    ax=ax1, title=r'Learning Curve of Loss Function CE')
gan_hist_df.loc[:, "KL Divergence"].plot(
    ax=ax2, title="Learning Curve of KL Divergence")
plt.show()
```



Observations

We notice that the model started collapsing after around 40 - 45 epochs as we can see the start of the divergence of g_loss and d_loss and the sudden drop in KL Divergence. From this, this shows that we will be using model epochs 40 to do a manual evaluation of the DCGAN model. We also note that by around 40 - 45 epochs the d_loss and g_loss starts to diverge from each other which suggest Model Collapsing occurred. This suggest that either the generator is too weak or the discriminator is too strong.

```
In [ ]: # Loading Weights for best Generator
saved_weights = 'GAN/generator-epoch-40.h5'
gan.generator.load_weights(saved_weights)
gan.generator.summary()
```

Model: "generator_GAN"

Layer (type)	Output Shape	Param #
=====		
dense_18 (Dense)	(None, 2048)	264192
reshape_7 (Reshape)	(None, 2, 2, 512)	0
conv2d_transpose_22 (Conv2D Transpose)	(None, 4, 4, 256)	2097408
batch_normalization_165 (BatchNormalization)	(None, 4, 4, 256)	1024
=====		
dense_18 (Dense)	(None, 2048)	264192
reshape_7 (Reshape)	(None, 2, 2, 512)	0
conv2d_transpose_22 (Conv2D Transpose)	(None, 4, 4, 256)	2097408
batch_normalization_165 (BatchNormalization)	(None, 4, 4, 256)	1024
p_re_lu_19 (PReLU)	(None, 4, 4, 256)	4096
conv2d_transpose_23 (Conv2D Transpose)	(None, 8, 8, 128)	524416
batch_normalization_166 (BatchNormalization)	(None, 8, 8, 128)	512
p_re_lu_20 (PReLU)	(None, 8, 8, 128)	8192
conv2d_transpose_24 (Conv2D Transpose)	(None, 16, 16, 64)	131136
batch_normalization_167 (BatchNormalization)	(None, 16, 16, 64)	256
p_re_lu_21 (PReLU)	(None, 16, 16, 64)	16384
conv2d_transpose_25 (Conv2D Transpose)	(None, 32, 32, 3)	3075
=====		
Total params:	3,050,691	
Trainable params:	3,049,795	
Non-trainable params:	896	

In []:

```
# Generating 1000 Synthetic Images
random_noise = tf.random.normal(shape=(1000, NOISE))
synthetic_images = gan.generator.predict(random_noise)
print("Latent Vector Dim: {} \t Generated Images Dim: {}".format(
    random_noise.shape, synthetic_images.shape))

# Scaling back to [0, 1]
synthetic_images -= -1
synthetic_images /= (1 - (-1))

# Display 15 randomly sampled images
fig = plt.figure(figsize=(10, 10), tight_layout=True)
for i in range(25):
    rand_idx = np.random.randint(0, len(synthetic_images))
```

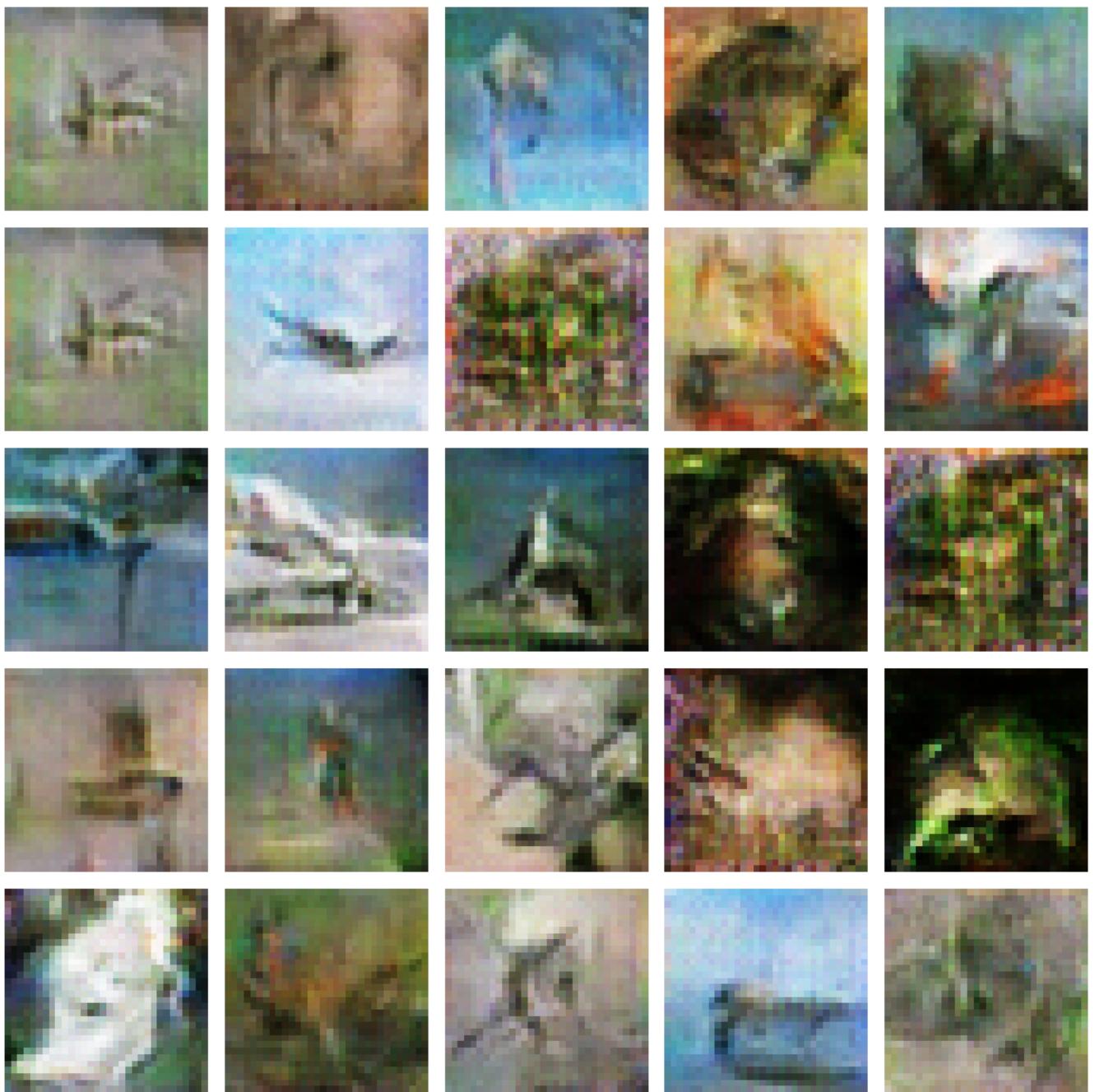
```

ax = fig.add_subplot(5, 5, i+1)
ax.imshow(synthetic_images[rand_idx])
ax.axis('off')
plt.show()

```

32/32 [=====] - 4s 10ms/step

Latent Vector Dim: (1000, 128) Generated Images Dim: (1000, 32, 32, 3)



Observations

We can see that the images are not clear and have a lot of different features of different class to help generate the images. It is hard to identify the type of images and which class it is trying to act like. Therefore models like Conditional GAN [cGAN] can help solve this issue.

```

In [ ]: gan_fid_class = GAN_FID(batch_size=512, noise=128,
                           sample_size=10000, buffer_size=1024)
gan_fid_class.fit(generator=gan.generator, train_data=x_train)
fid_score = gan_fid_class.evaluate()

```

Computing Real Image Embeddings
0% | 0/20 [00:00<?, ?it/s]

```

16/16 [=====] - 9s 153ms/step
16/16 [=====] - 3s 162ms/step
16/16 [=====] - 3s 160ms/step
16/16 [=====] - 3s 160ms/step
16/16 [=====] - 3s 159ms/step
16/16 [=====] - 3s 159ms/step
16/16 [=====] - 2s 154ms/step
16/16 [=====] - 2s 156ms/step
16/16 [=====] - 3s 160ms/step
16/16 [=====] - 2s 136ms/step
16/16 [=====] - 2s 139ms/step
16/16 [=====] - 2s 139ms/step
16/16 [=====] - 2s 139ms/step
16/16 [=====] - 2s 140ms/step
16/16 [=====] - 2s 137ms/step
16/16 [=====] - 2s 135ms/step
16/16 [=====] - 2s 141ms/step
16/16 [=====] - 2s 140ms/step
16/16 [=====] - 2s 138ms/step
16/16 [=====] - 2s 141ms/step
Computing Generated Image Embeddings
 0% | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 2s 141ms/step
16/16 [=====] - 2s 140ms/step
16/16 [=====] - 2s 141ms/step
16/16 [=====] - 2s 137ms/step
16/16 [=====] - 2s 141ms/step
16/16 [=====] - 2s 143ms/step
16/16 [=====] - 2s 142ms/step
16/16 [=====] - 2s 140ms/step
16/16 [=====] - 2s 142ms/step
16/16 [=====] - 2s 145ms/step
16/16 [=====] - 2s 141ms/step
16/16 [=====] - 2s 143ms/step
16/16 [=====] - 2s 143ms/step
16/16 [=====] - 2s 153ms/step
16/16 [=====] - 3s 168ms/step
16/16 [=====] - 2s 156ms/step
16/16 [=====] - 2s 153ms/step
16/16 [=====] - 2s 149ms/step
16/16 [=====] - 2s 149ms/step
16/16 [=====] - 2s 151ms/step

```

Computed Embeddings Real Images Embedding Shape: (10240, 2048) Generated Images Embe
dding Shape: (10240, 2048)

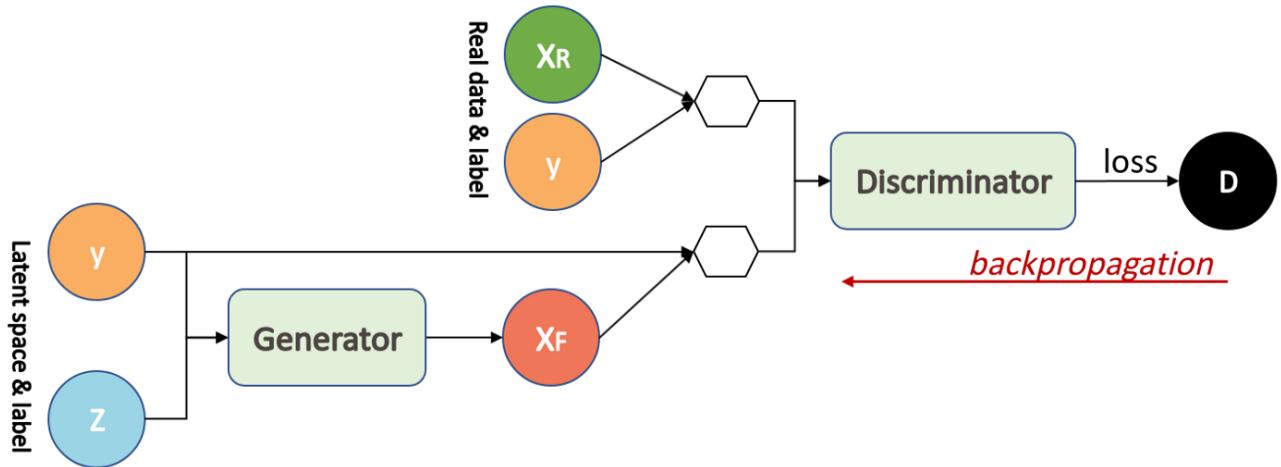
The computed FID score is: 153.4580126583096

Observations

We see that the FID score is 153.4580126583096. This high FID score shows that the generated images are significantly different from the real images. Therefore more changes should be done to improve the FID score like using regularisation techniques or different architectures.

cGAN

cGAN also known as Conditional GAN allows the GAN model to control what type of images are generated. Therefore, in the cGAN model, labels will need to be parse into the model.



cGAN Discriminator

Basically, the discriminator is a binary classifier. For the cGAN model, we will make a typical CNN network. However, unlike a typical CNN network, we will be modifying it such that cGAN model will be improved.

1. Pooling Layers in a classic CNN are used to down sample activation features maps. However, this is not recommended for GAN as information will be lost. Therefore, we will be using a 2×2 Stride Convolution
2. Due to the vanishing gradient problem, the activation function for a GAN is usually a LeakyReLU. It allows gradients to flow easier through the architecture. This allows the function to compute the greatest value between the features and a small factor.
3. As sparse gradients will occur and diminish over time [random spikes in the loss during training], there will be many numerical operations performed on them. To fix this, we will add BatchNormalization to the network so that it can reduce covariate shifts. BatchNormalization will be used before non-Gaussian activations like LeakyReLU.

```
In [ ]: # function to create discriminator model
def create_cGAN_discriminator(image_size):
    # input image
    img_input = Input(shape=(image_size), name='Image_Input')
    # conditions y
    conditions = Input(shape=(10,), name='Conditions_y')
    base_discriminator = Sequential([
        Conv2D(64, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
        Conv2D(128, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
        Conv2D(256, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
        Conv2D(512, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
    ], name='Base_Discriminator')
    discriminator = base_discriminator(img_input)

    discriminator = GlobalAveragePooling2D()(discriminator)

    # Concatenate - combine with conditions y
    merged_layer = Concatenate()([discriminator, conditions])
    discriminator = Dense(512, activation='relu')(merged_layer)

    # Output
```

```

discriminator = Dense(1, activation='sigmoid',
                     name='Output_Layer')(discriminator)

discriminator = Model(inputs=[img_input, conditions],
                      outputs=discriminator, name='discriminator_cGAN')
return discriminator

```

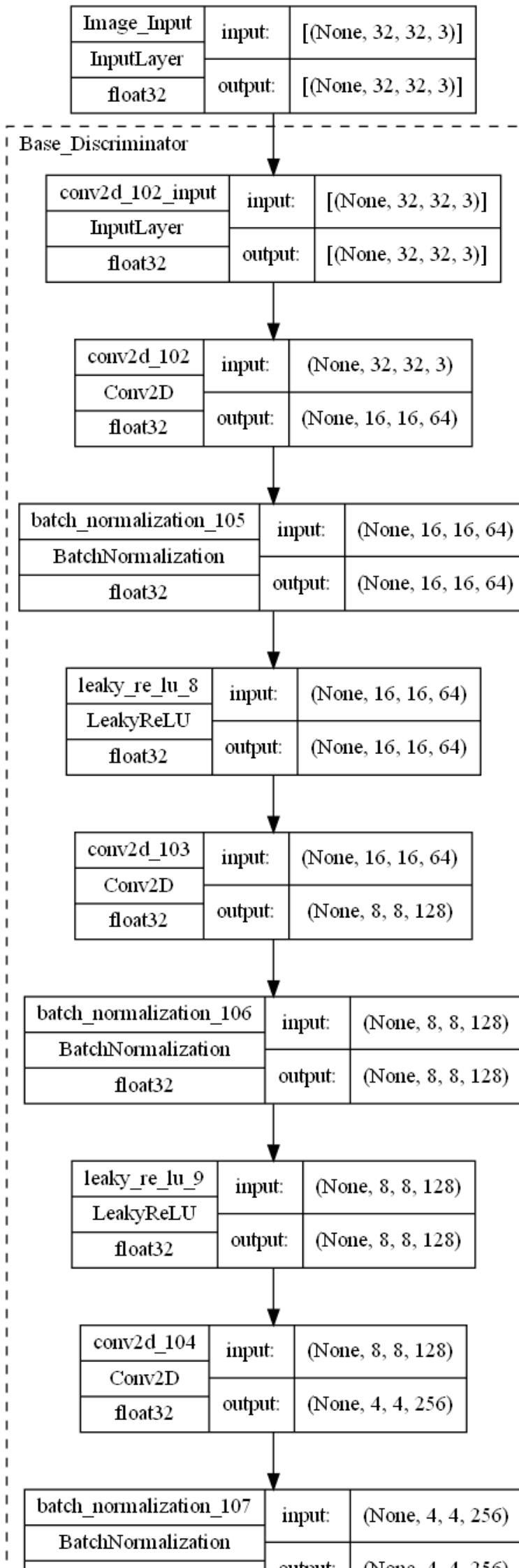
```
create_cGAN_discriminator(image_size=IMG_SIZE).summary()
```

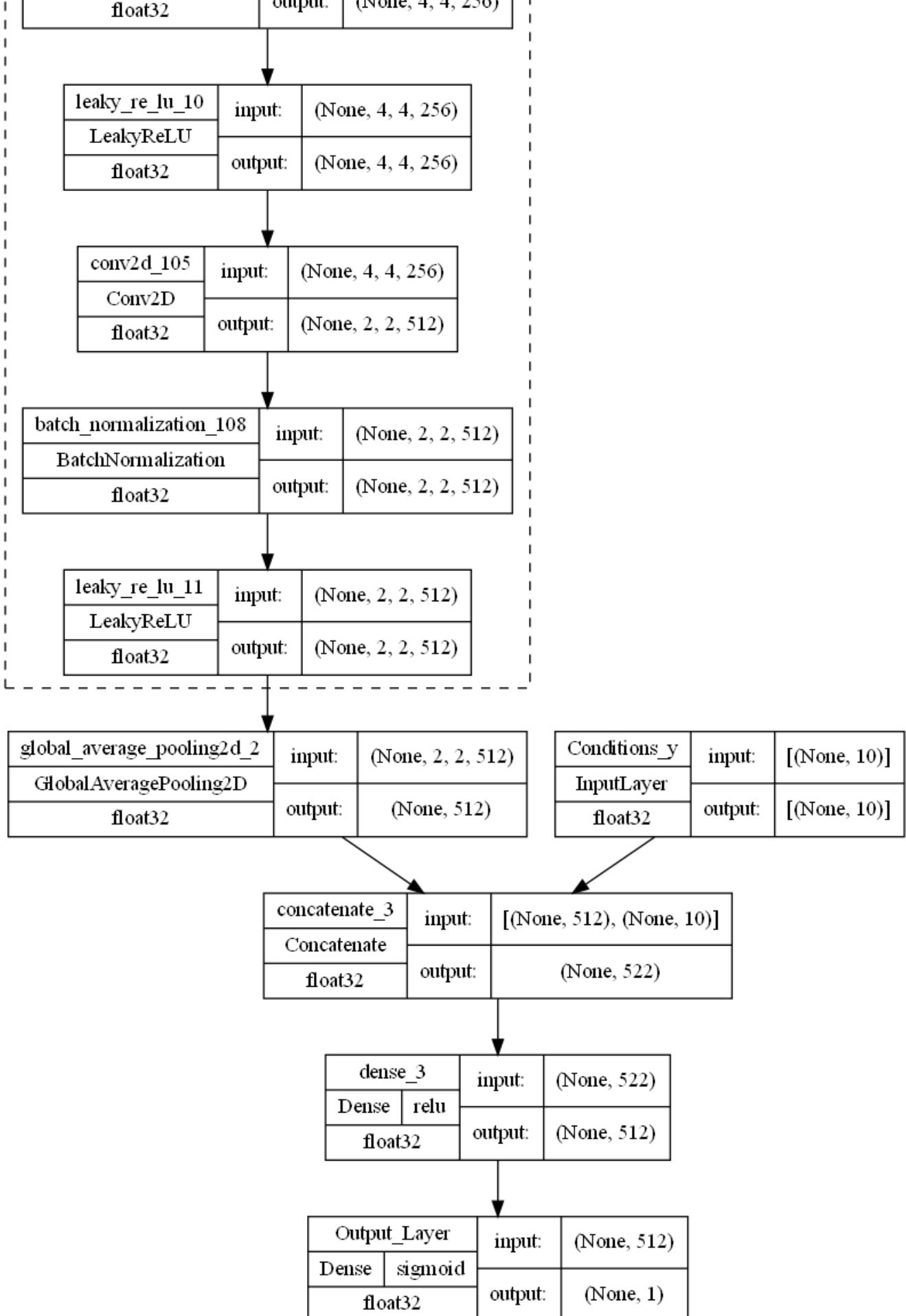
Model: "discriminator_cGAN"

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
Image_Input (InputLayer)	[(None, 32, 32, 3)]	0	[]
Base_Discriminator (Sequential (None, 2, 2, 512))	2760384		['Image_Input[0][0]']
global_average_pooling2d_7 (G1 (None, 512))	0		['Base_Discriminator[0][0]']
<hr/>			
Layer (type)	Output Shape	Param #	Connected to
<hr/>			
Image_Input (InputLayer)	[(None, 32, 32, 3)]	0	[]
Base_Discriminator (Sequential (None, 2, 2, 512))	2760384		['Image_Input[0][0]']
global_average_pooling2d_7 (G1 (None, 512))	0		['Base_Discriminator[0][0]']
globalAveragePooling2D			
Conditions_y (InputLayer)	[(None, 10)]	0	[]
concatenate_12 (Concatenate)	(None, 522)	0	['global_average_pooling2d_7[0][0]
]',
dense_10 (Dense)	(None, 512)	267776	['concatenate_12[0][0]']
Output_Layer (Dense)	(None, 1)	513	['dense_10[0][0]']
<hr/>			
<hr/>			
Total params: 3,028,673			
Trainable params: 3,026,753			
Non-trainable params: 1,920			

In []: plot_model(create_cGAN_discriminator(image_size=IMG_SIZE), to_file="images/models/cGAN_discriminator_cGAN", show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True)

Out[]:





cGAN Generator

Unlike your typical neural architectures, the generator will have multiple convolutional upscaled.

1. Transposed Convolution Layers allows the normal convolution to go in the opposite direction. The main purpose is to allow the CNN to upsample from the noise latent vector to generated the images.

2. As sparse gradients will occur and diminish over time [random spikes in the loss during training], there will be many numerical operations performed on them. To fix this, we will add BatchNormalization to the network so that it can reduce covariate shifts.
3. Due to the sparse gradient during training and effects like the vanishing gradient problems, we can mitigate this effects using Leaky ReLU.

```
In [ ]: # function to create generator model
def create_cGAN_generator(noise):
    # Latent noise vector z
    z = Input(shape=(noise,), name="Latent_Noise_Vector_z")

    # conditions y
    conditions = Input(shape=(10,), name='Conditions_y')

    # Generator network
    merged_layer = Concatenate()([z, conditions])

    # FC: 2x2x512
    generator = Dense(2*2*512, activation='relu')(merged_layer)
    generator = BatchNormalization(momentum=0.9)(generator)
    generator = LeakyReLU(alpha=0.1)(generator)
    generator = Reshape((2, 2, 512))(generator)

    base_generator = Sequential([
        # Conv 1: 4x4x256
        Conv2DTranspose(256, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
        # Conv 2: 8x8x128
        Conv2DTranspose(128, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
        # Conv 3: 16x16x64
        Conv2DTranspose(64, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
    ], name='Base_Generator')
    generator = base_generator(generator)

    # Conv 4: 32x32x3
    generator = Conv2DTranspose(3, kernel_size=4, strides=2, padding='same',
                               activation='tanh', name='Output_Layer')(generator)

    generator = Model(inputs=[z, conditions],
                      outputs=generator, name='generator_cGAN')
    return generator

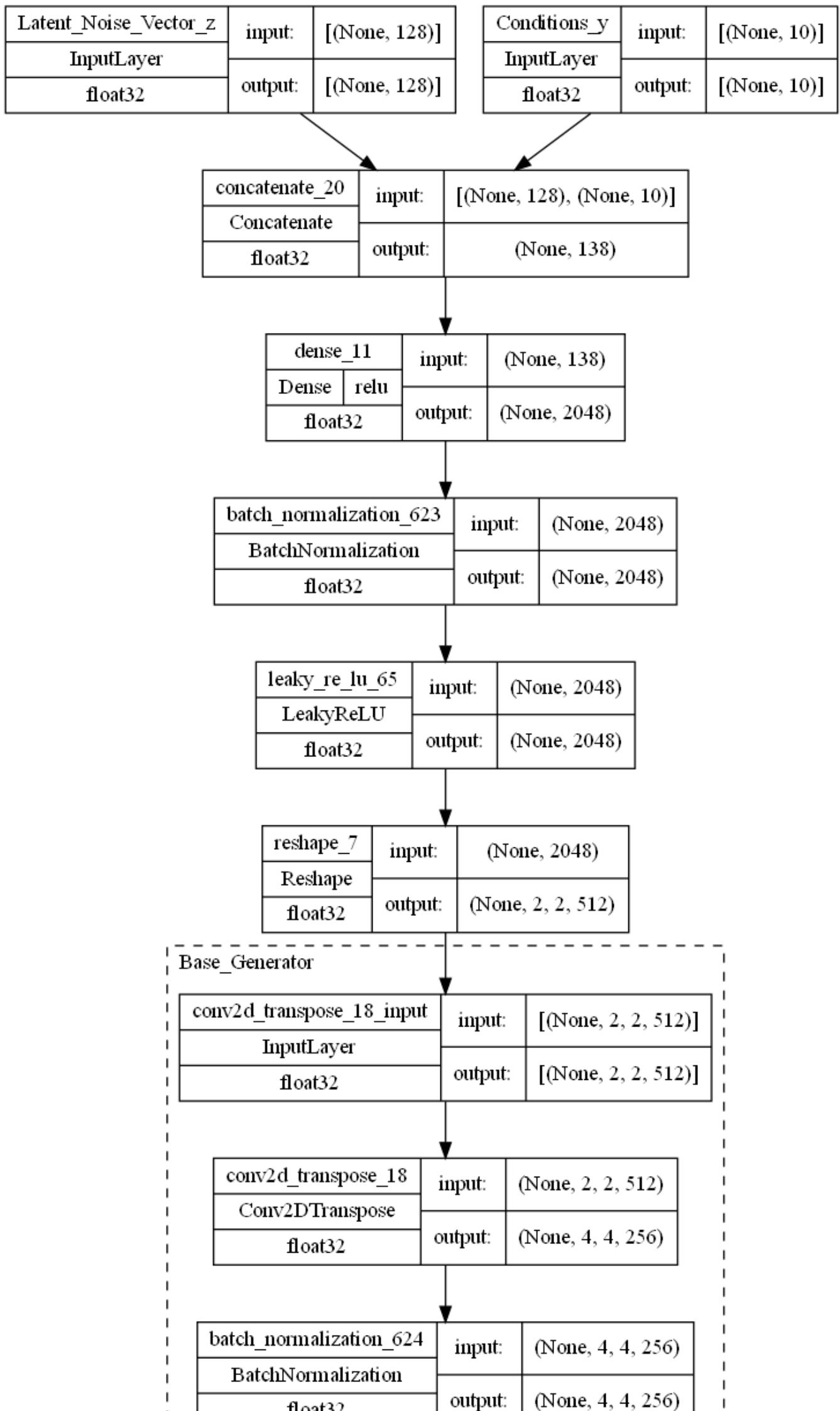
create_cGAN_generator(noise=NOISE).summary()
```

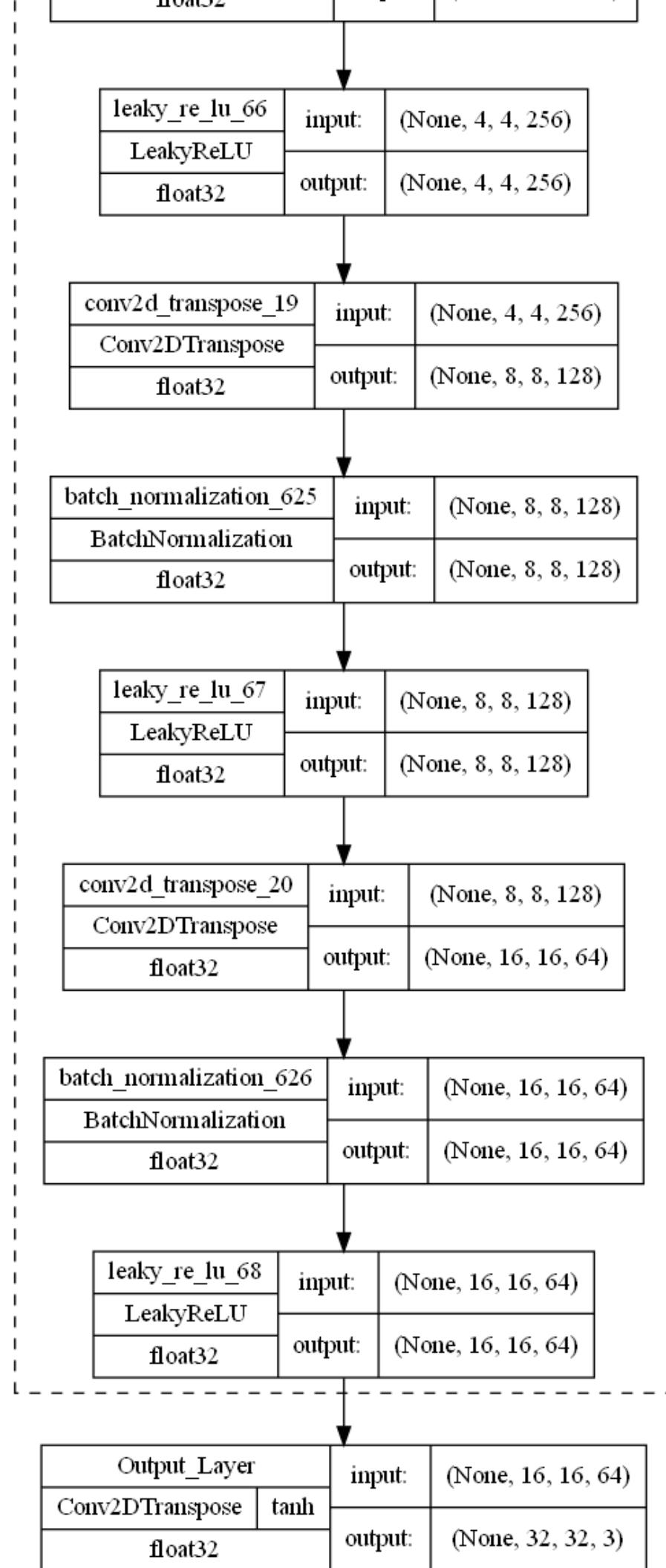
Model: "generator_cGAN"

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
<hr/>			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_13 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_11 (Dense)	(None, 2048)	284672	['concatenate_13[0][0]']
batch_normalization_138 (Batch Normalization)	(None, 2048)	8192	['dense_11[0][0]']
leaky_re_lu_28 (LeakyReLU)	(None, 2048)	0	['batch_normalization_138[0][0]']
reshape_4 (Reshape)	(None, 2, 2, 512)	0	['leaky_re_lu_28[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2754752	['reshape_4[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
<hr/>			
=====			
Total params: 3,050,691			
Trainable params: 3,045,699			
Non-trainable params: 4,992			

In []: plot_model(create_cGAN_generator(noise=NOISE), to_file="images/models/cGAN_generator.png", show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True)

Out[]:





It is similar to the GAN Training process but with the inclusion of labels. To help us with the code, we will be referring to Keras example on a conditional GAN.

https://github.com/keras-team/keras-io/blob/master/examples/generative/conditional_gan.py

In []:

```
class ConditionalGAN(tf.keras.Model):
    def __init__(self, discriminator, generator, noise):
        super(ConditionalGAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.noise = noise
        self.gen_loss_tracker = tf.keras.metrics.Mean(name="generator_loss")
        self.disc_loss_tracker = tf.keras.metrics.Mean(
            name="discriminator_loss")
        self.d_xy_tracker = tf.keras.metrics.Mean(name='Mean D(x|y)')
        self.d_g_zy_tracker = tf.keras.metrics.Mean(name='Mean D(G(z|y))')
        self.kl = tf.keras.metrics.KLDivergence()

    def compile(self, d_optimizer, g_optimizer, loss_fn):
        super(ConditionalGAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.loss_fn = loss_fn

    def train_step(self, data):
        ### TRAINING DISCRIMINATOR ###
        # Unpack the data.
        real_images, condition = data

        # Sample for Latent noise vector z
        batch_size = tf.shape(real_images)[0]
        latent_noise_vector = tf.random.normal(shape=(batch_size, self.noise))

        # Maps the noise Latent vector and labels to generate fake images.
        generated_images = self.generator([latent_noise_vector, condition])

        # Combine with real images
        combined_images = tf.concat([generated_images, real_images], axis=0)
        combined_condition = tf.concat([condition, condition], axis=0)

        # Discrimination
        labels = tf.concat([
            tf.zeros((batch_size, 1)), tf.ones((batch_size, 1))], axis=0
        )

        # Train the discriminator.
        with tf.GradientTape() as tape:
            first_predictions = self.discriminator(
                [combined_images, combined_condition])
            d_loss = self.loss_fn(labels, first_predictions)
        grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights)
        )

        # Computing D(x/y)
        d_xy = tf.math.reduce_mean(first_predictions)

        ### TRAINING GENERATOR ###
        latent_noise_vector = tf.random.normal(shape=(batch_size, self.noise))

        # Assemble labels that say "all real images".
        misleading_labels = tf.ones((batch_size, 1))
```

```

    with tf.GradientTape() as tape:
        fake_images = self.generator([latent_noise_vector, condition])
        second_predictions = self.discriminator([fake_images, condition])
        g_loss = self.loss_fn(misleading_labels, second_predictions)
        grads = tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(
            zip(grads, self.generator.trainable_weights))

    # Computing D(G(z|y))
    d_g_zy = tf.math.reduce_mean(second_predictions)

    # Monitor loss and metrics.
    self.gen_loss_tracker.update_state(g_loss)
    self.disc_loss_tracker.update_state(d_loss)
    self.d_xy_tracker.update_state(d_xy)
    self.d_g_zy_tracker.update_state(d_g_zy)
    self.kl.update_state(real_images, fake_images)

    return {
        "d_loss": self.disc_loss_tracker.result(),
        "g_loss": self.gen_loss_tracker.result(),
        "D(x|y)": self.d_xy_tracker.result(),
        "D(G(z|y))": self.d_g_zy_tracker.result(),
        "KL Divergence": self.
        kl.result(),
    }
}

```

cGAN Callback

To help monitor the cGAN's loss and progress and visualise the images after every patience. We will need to make a custom tensorflow callback that will help monitor the cGAN.

```

In [ ]: class cGANMonitor(keras.callbacks.Callback):
    def __init__(self, num_img=20, noise=128, patience=10, vmin=0, vmax=1):
        self.num_img = num_img
        self.noise = noise
        self.patience = patience
        self.vmin = vmin
        self.vmax = vmax
        self.latent_noise_vector = tf.random.normal(
            shape=(self.num_img, self.noise))
        self.conditions = to_categorical([
            0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
            0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

    def generate_plot(self):
        # Generate Images
        generated_images = self.model.generator.predict(
            [self.latent_noise_vector, self.conditions])
        # Normalise Image from [vmin, vmax] to [0, 1]
        generated_images -= self.vmin
        generated_images /= (self.vmax - self.vmin)
        row_size = int(np.ceil(self.num_img/5))
        fig = plt.figure(figsize=(10, 2*row_size), tight_layout=True)
        for i in range(self.num_img):
            ax = fig.add_subplot(row_size, 5, i+1)
            ax.imshow(generated_images[i])
            ax.set_title(class_labels[i % 10])
            ax.axis('off')
        plt.show()

    def save_weights(self, epoch=None):
        try:
            if epoch != None:
                name = 'cGAN/generator-epoch-{}.h5'.format(epoch)
                print('Generator Checkpoint - {}'.format(name))

```

```

        self.model.generator.save_weights(
            filepath=name,
            save_format='h5'
        )
    except Exception as e:
        print(e)

    def on_epoch_begin(self, epoch, logs=None):
        if epoch % self.patience == 0:
            self.generate_plot()
            self.save_weights(epoch)

    def on_train_end(self, epoch, logs=None):
        self.generate_plot()
        self.save_weights('Full Train')

```

```
In [ ]: callbacks = [
    cGANMonitor(num_img=20, noise=128, patience=5, vmin=-1, vmax=1),
]
```

cGAN Training

After setting the cGAN architecture, we can now run the cGAN models to generate images. First, we need to convert the data into a tensorflow dataset and training using multiprocessing to speed up process.

```

In [ ]: tf.keras.backend.clear_session()
K.clear_session()

cond_gan = ConditionalGAN(
    discriminator=create_cGAN_discriminator(image_size=IMG_SIZE),
    generator=create_cGAN_generator(noise=NOISE),
    noise=NOISE
)

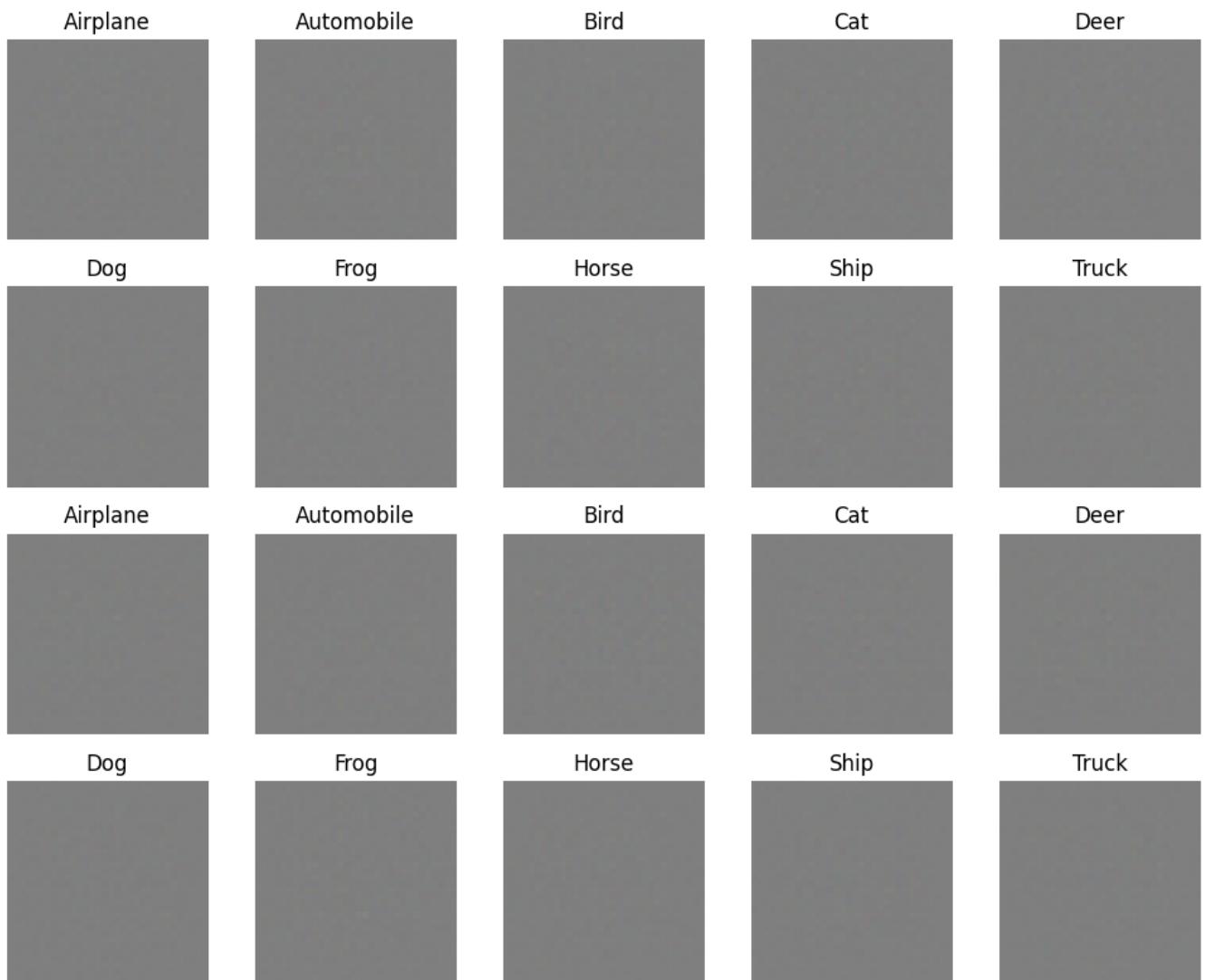
cond_gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    loss_fn=keras.losses.BinaryCrossentropy(),
)

dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
dataset = dataset.shuffle(buffer_size=1024).batch(
    BATCH_SIZE, num_parallel_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)

cond_gan_hist = cond_gan.fit(
    dataset, epochs=200, use_multiprocessing=True, workers=16, callbacks=callbacks)

```

```
1/1 [=====] - 0s 355ms/step
```



Generator Checkpoint - cGAN/generator-epoch-0.h5

Epoch 1/200

782/782 [=====] - 65s 79ms/step - d_loss: 0.3290 - g_loss: 2.8474 - D(x|y): 0.5407 - D(G(z|y)): 0.1412 - KL Divergence: 6.1464

Epoch 2/200

782/782 [=====] - 66s 84ms/step - d_loss: 0.2402 - g_loss: 3.7414 - D(x|y): 0.4965 - D(G(z|y)): 0.0524 - KL Divergence: 5.9927

Epoch 3/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2934 - g_loss: 2.8933 - D(x|y): 0.4990 - D(G(z|y)): 0.0857 - KL Divergence: 4.8510

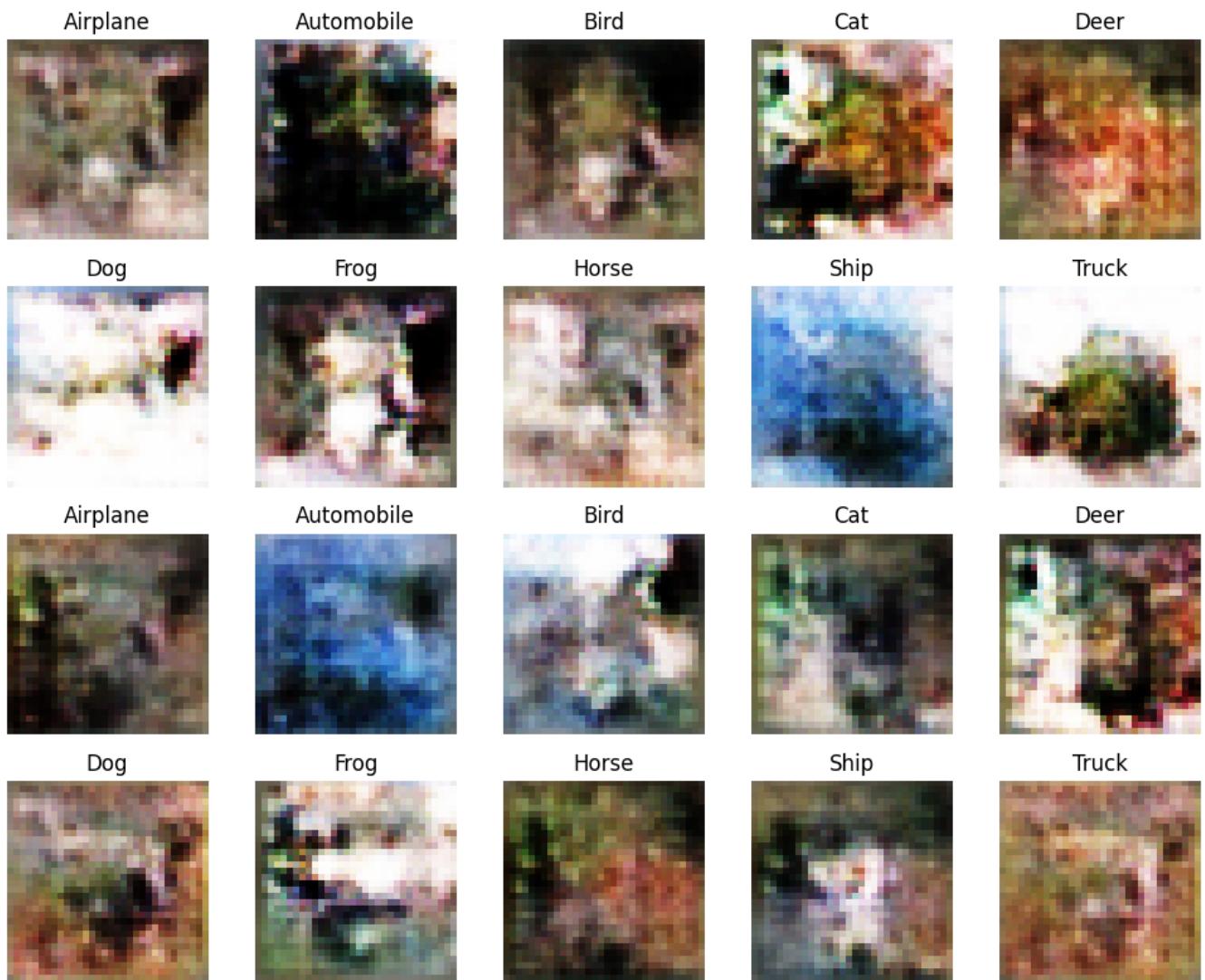
Epoch 4/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2417 - g_loss: 3.2650 - D(x|y): 0.5028 - D(G(z|y)): 0.0628 - KL Divergence: 4.4427

Epoch 5/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2386 - g_loss: 3.5025 - D(x|y): 0.5005 - D(G(z|y)): 0.0572 - KL Divergence: 4.5276

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - cGAN/generator-epoch-5.h5

Epoch 6/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2520 - g_loss: 3.2240 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0781 - KL Divergence: 5.1005

Epoch 7/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2711 - g_loss: 3.0058 -
 $D(x|y)$: 0.4993 - $D(G(z|y))$: 0.0978 - KL Divergence: 4.7024

Epoch 8/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2723 - g_loss: 2.9330 -
 $D(x|y)$: 0.5011 - $D(G(z|y))$: 0.1089 - KL Divergence: 4.7046

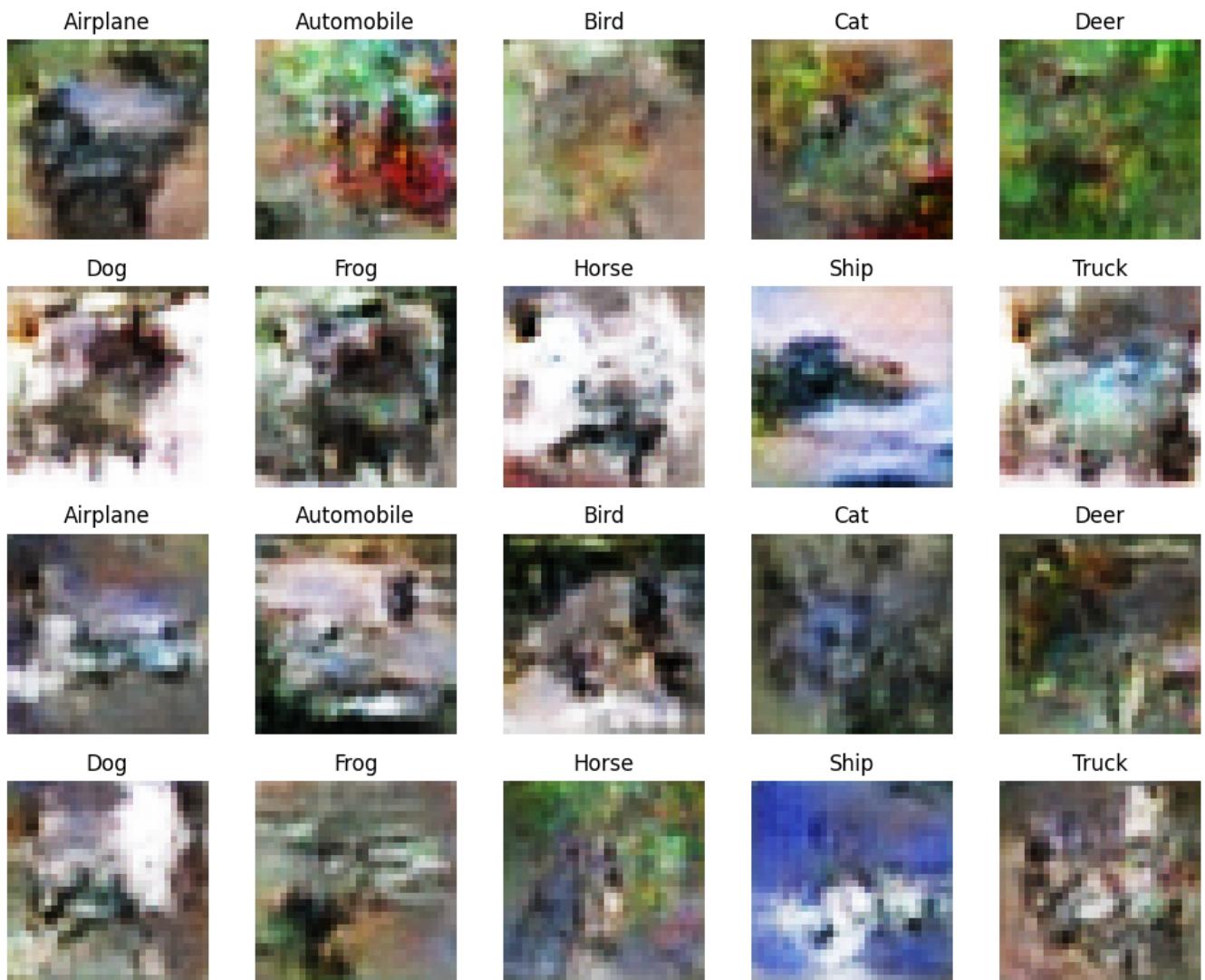
Epoch 9/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2756 - g_loss: 2.8864 -
 $D(x|y)$: 0.5011 - $D(G(z|y))$: 0.1193 - KL Divergence: 4.8275

Epoch 10/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2889 - g_loss: 2.6858 -
 $D(x|y)$: 0.5009 - $D(G(z|y))$: 0.1370 - KL Divergence: 4.7402

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - cGAN/generator-epoch-10.h5

Epoch 11/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.3023 - g_loss: 2.6386 - D(x|y): 0.5003 - D(G(z|y)): 0.1455 - KL Divergence: 4.7275

Epoch 12/200

782/782 [=====] - 65s 82ms/step - d_loss: 0.2889 - g_loss: 2.7427 - D(x|y): 0.4999 - D(G(z|y)): 0.1397 - KL Divergence: 4.7474

Epoch 13/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.2824 - g_loss: 2.7091 - D(x|y): 0.5006 - D(G(z|y)): 0.1464 - KL Divergence: 4.7949

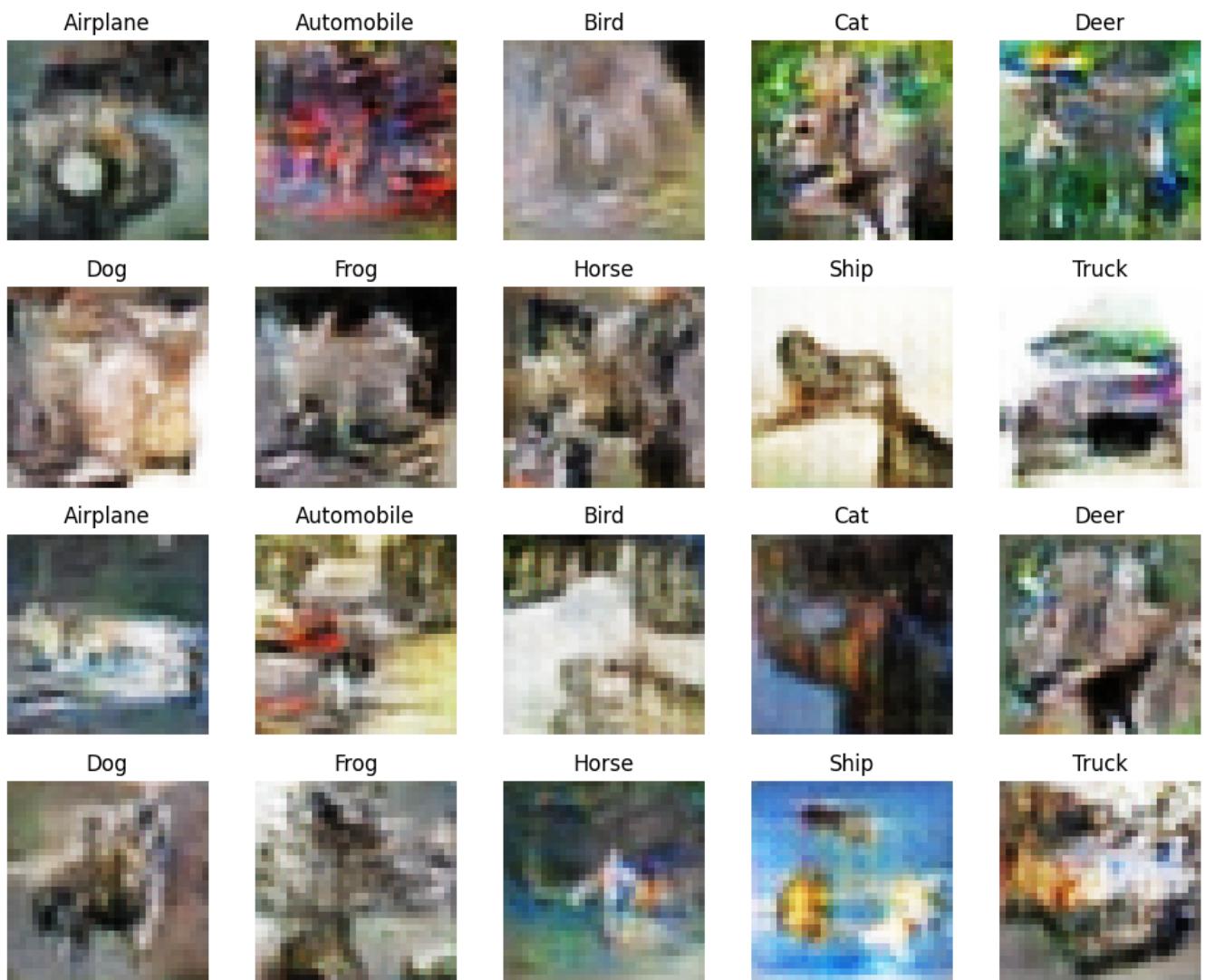
Epoch 14/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2738 - g_loss: 2.7683 - D(x|y): 0.5006 - D(G(z|y)): 0.1444 - KL Divergence: 4.7494

Epoch 15/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.2598 - g_loss: 2.8938 - D(x|y): 0.5011 - D(G(z|y)): 0.1375 - KL Divergence: 4.7508

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - cGAN/generator-epoch-15.h5

Epoch 16/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.2525 - g_loss: 2.9388 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.1374 - KL Divergence: 4.6120

Epoch 17/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.2284 - g_loss: 3.1960 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.1217 - KL Divergence: 4.5727

Epoch 18/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.2133 - g_loss: 3.3671 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.1135 - KL Divergence: 4.4471

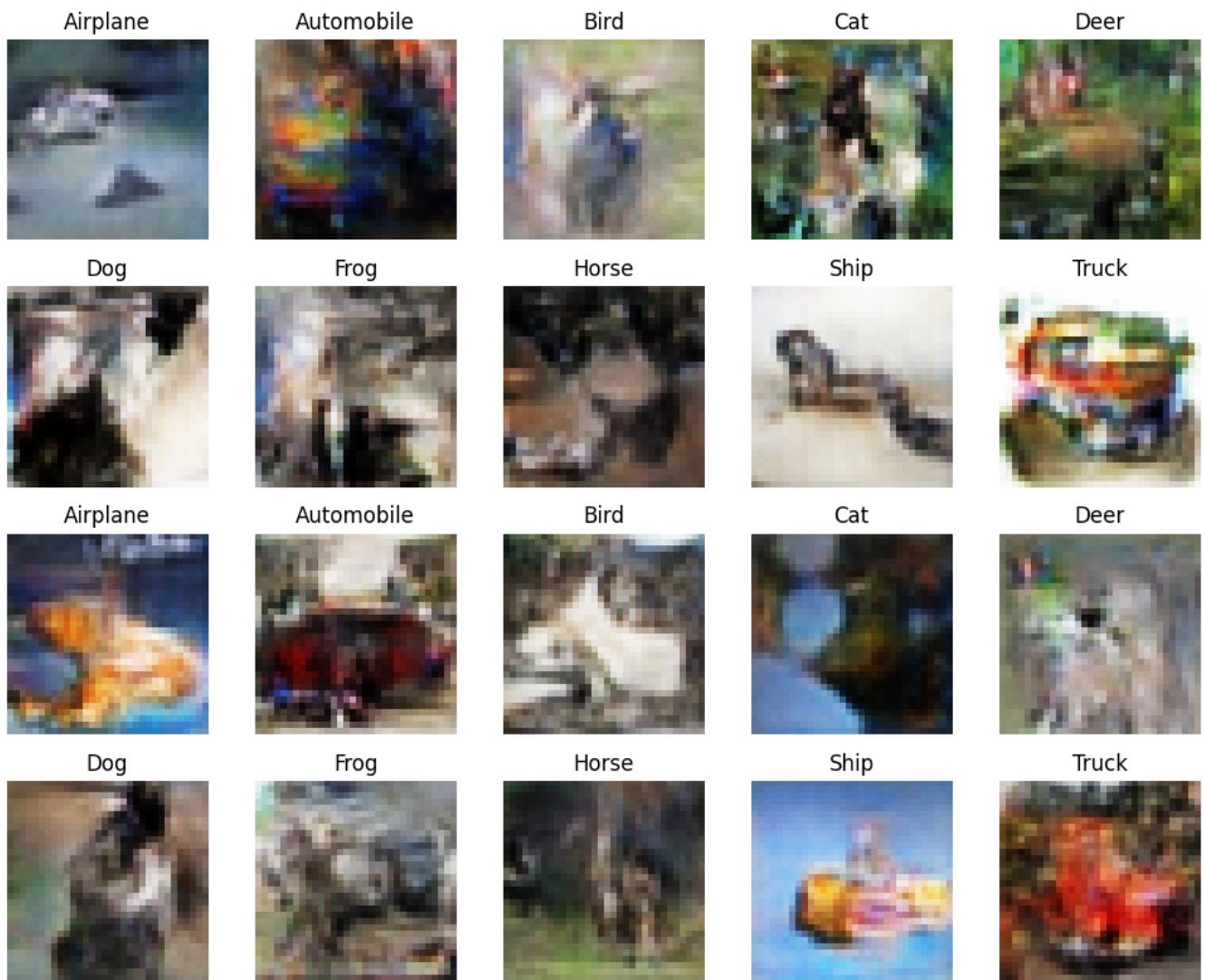
Epoch 19/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.2055 - g_loss: 3.4754 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.1114 - KL Divergence: 4.5915

Epoch 20/200

782/782 [=====] - 61s 78ms/step - d_loss: 0.1963 - g_loss: 3.6419 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.1049 - KL Divergence: 4.5955

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - cGAN/generator-epoch-20.h5

Epoch 21/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1920 - g_loss: 3.7617 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.1020 - KL Divergence: 4.5135

Epoch 22/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1781 - g_loss: 3.8728 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0979 - KL Divergence: 4.5717

Epoch 23/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.1659 - g_loss: 4.1535 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0881 - KL Divergence: 4.5492

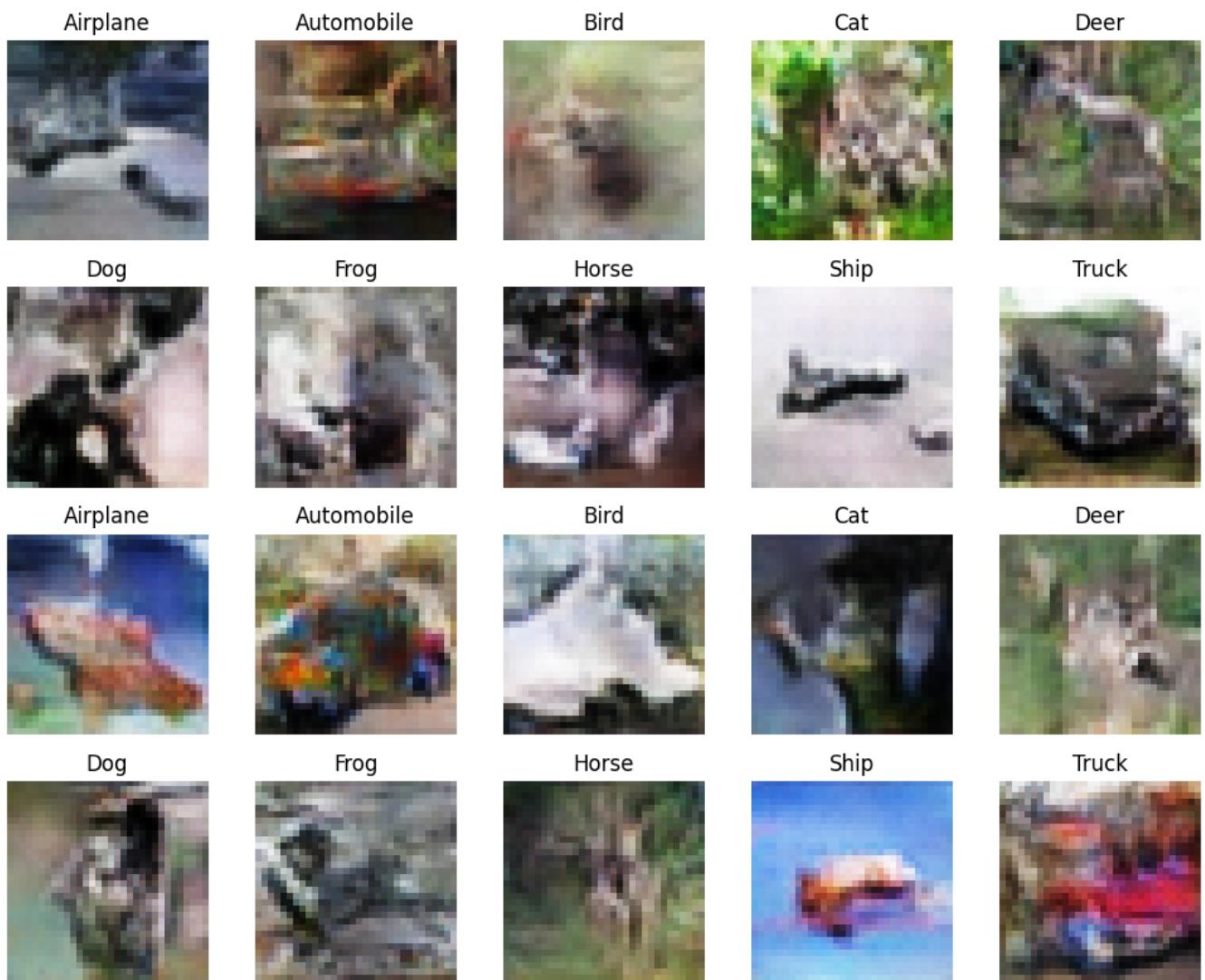
Epoch 24/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1670 - g_loss: 4.1812 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0890 - KL Divergence: 4.7242

Epoch 25/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.1561 - g_loss: 4.3943 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.0861 - KL Divergence: 4.7061

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - cGAN/generator-epoch-25.h5

Epoch 26/200

782/782 [=====] - 72s 93ms/step - d_loss: 0.1571 - g_loss: 4.4908 - D(x|y): 0.5005 - D(G(z|y)): 0.0841 - KL Divergence: 4.6588

Epoch 27/200

782/782 [=====] - 78s 100ms/step - d_loss: 0.1483 - g_loss: 4.6073 - D(x|y): 0.5001 - D(G(z|y)): 0.0808 - KL Divergence: 4.6768

Epoch 28/200

782/782 [=====] - 84s 107ms/step - d_loss: 0.1423 - g_loss: 4.7443 - D(x|y): 0.5005 - D(G(z|y)): 0.0796 - KL Divergence: 4.6502

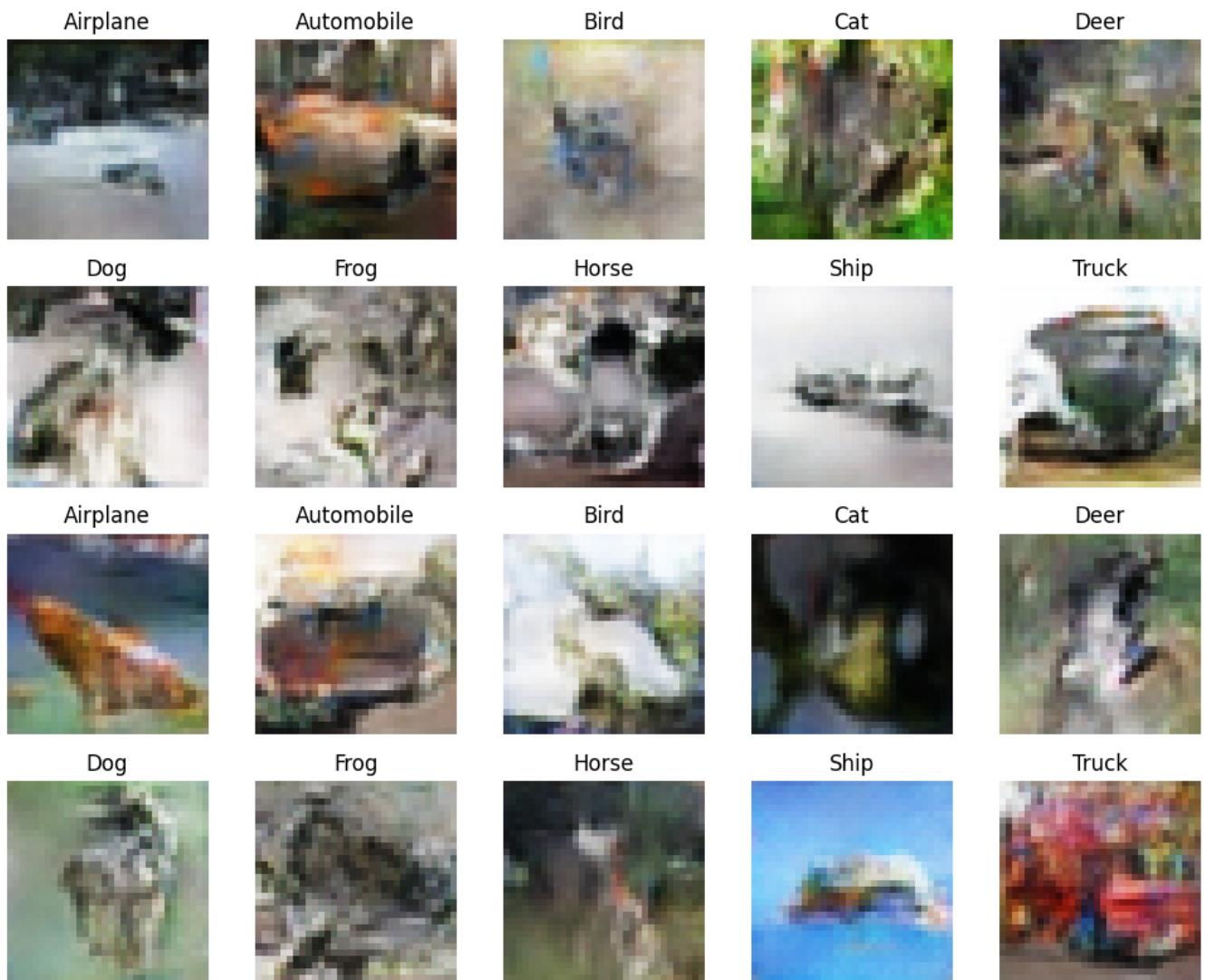
Epoch 29/200

782/782 [=====] - 70s 90ms/step - d_loss: 0.1425 - g_loss: 4.8908 - D(x|y): 0.4995 - D(G(z|y)): 0.0765 - KL Divergence: 4.6786

Epoch 30/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.1297 - g_loss: 5.0276 - D(x|y): 0.5003 - D(G(z|y)): 0.0731 - KL Divergence: 4.6257

1/1 [=====] - 0s 37ms/step



Generator Checkpoint - cGAN/generator-epoch-30.h5

Epoch 31/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1292 - g_loss: 5.0845 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0723 - KL Divergence: 4.6513

Epoch 32/200

782/782 [=====] - 74s 95ms/step - d_loss: 0.1969 - g_loss: 4.6288 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0967 - KL Divergence: 4.7050

Epoch 33/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.1264 - g_loss: 5.3292 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.0679 - KL Divergence: 4.6659

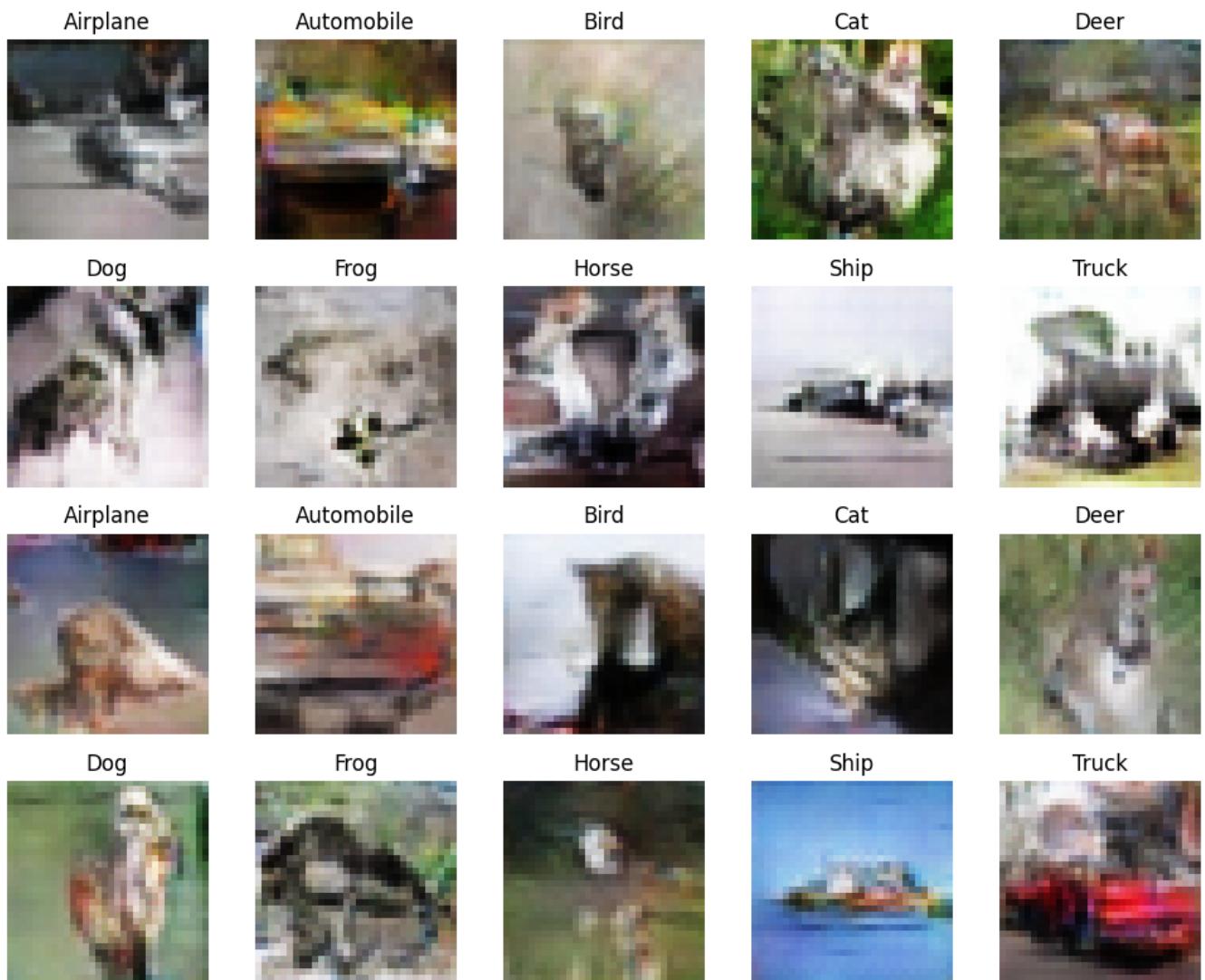
Epoch 34/200

782/782 [=====] - 79s 101ms/step - d_loss: 0.1197 - g_loss: 5.4663 -
 $D(x|y)$: 0.4996 - $D(G(z|y))$: 0.0668 - KL Divergence: 4.6905

Epoch 35/200

782/782 [=====] - 79s 101ms/step - d_loss: 0.1204 - g_loss: 5.5536 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0642 - KL Divergence: 4.6716

1/1 [=====] - 0s 31ms/step



Generator Checkpoint - cGAN/generator-epoch-35.h5

Epoch 36/200

782/782 [=====] - 79s 101ms/step - d_loss: 0.1108 - g_loss: 5.5071 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0636 - KL Divergence: 4.5958

Epoch 37/200

782/782 [=====] - 78s 99ms/step - d_loss: 0.1098 - g_loss: 5.8188 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0608 - KL Divergence: 4.7379

Epoch 38/200

782/782 [=====] - 78s 99ms/step - d_loss: 0.1080 - g_loss: 5.9050 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.0562 - KL Divergence: 4.6329

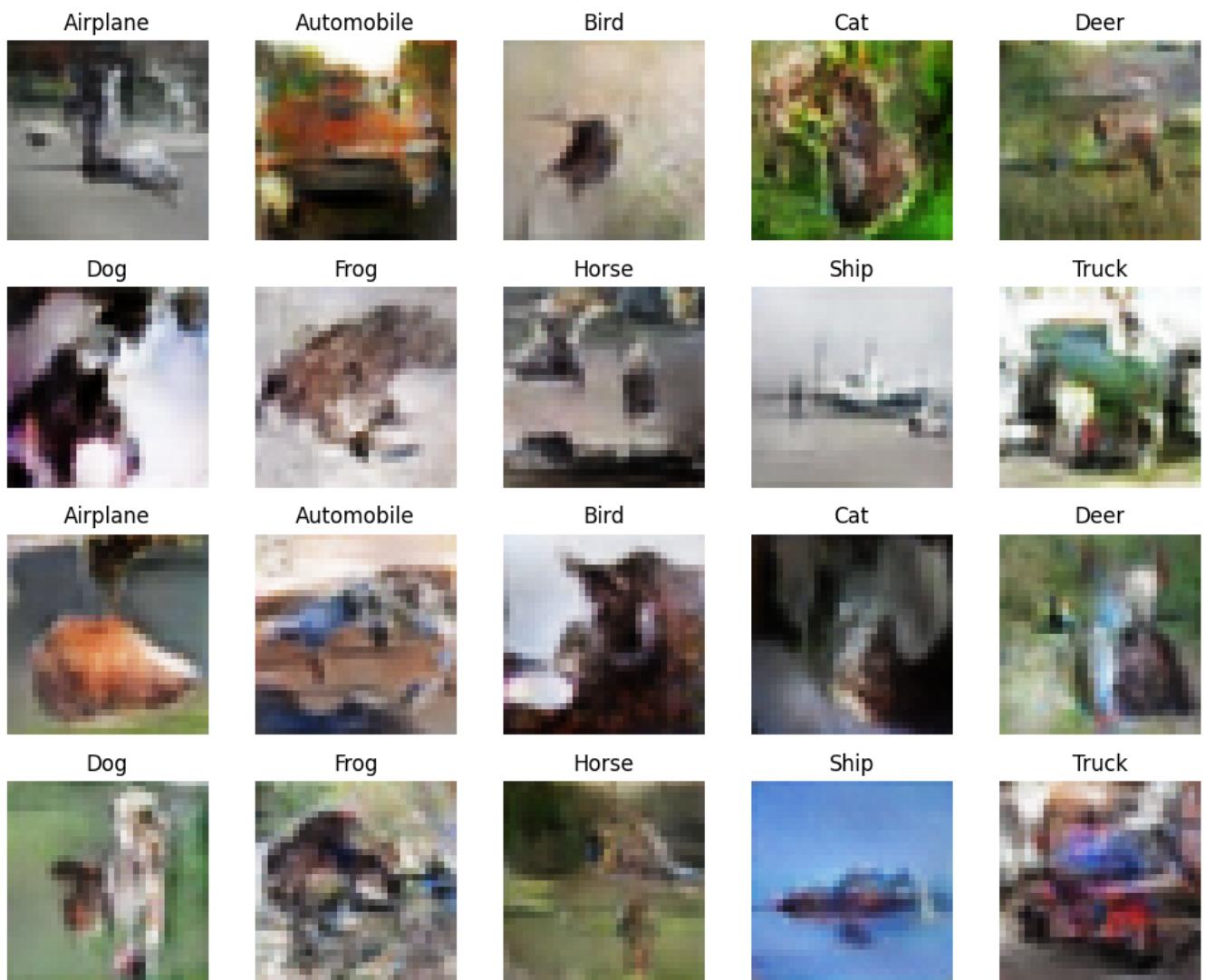
Epoch 39/200

782/782 [=====] - 79s 101ms/step - d_loss: 0.1068 - g_loss: 6.0401 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0576 - KL Divergence: 4.4598

Epoch 40/200

782/782 [=====] - 74s 95ms/step - d_loss: 0.1046 - g_loss: 5.9828 -
 $D(x|y)$: 0.5007 - $D(G(z|y))$: 0.0591 - KL Divergence: 4.6524

1/1 [=====] - 0s 29ms/step



Generator Checkpoint - cGAN/generator-epoch-40.h5

Epoch 41/200

782/782 [=====] - 71s 90ms/step - d_loss: 0.1025 - g_loss: 6.0877 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0571 - KL Divergence: 4.5610

Epoch 42/200

782/782 [=====] - 74s 94ms/step - d_loss: 0.1025 - g_loss: 6.2132 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0581 - KL Divergence: 4.6170

Epoch 43/200

782/782 [=====] - 97s 124ms/step - d_loss: 0.1008 - g_loss: 6.3210 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0544 - KL Divergence: 4.5913

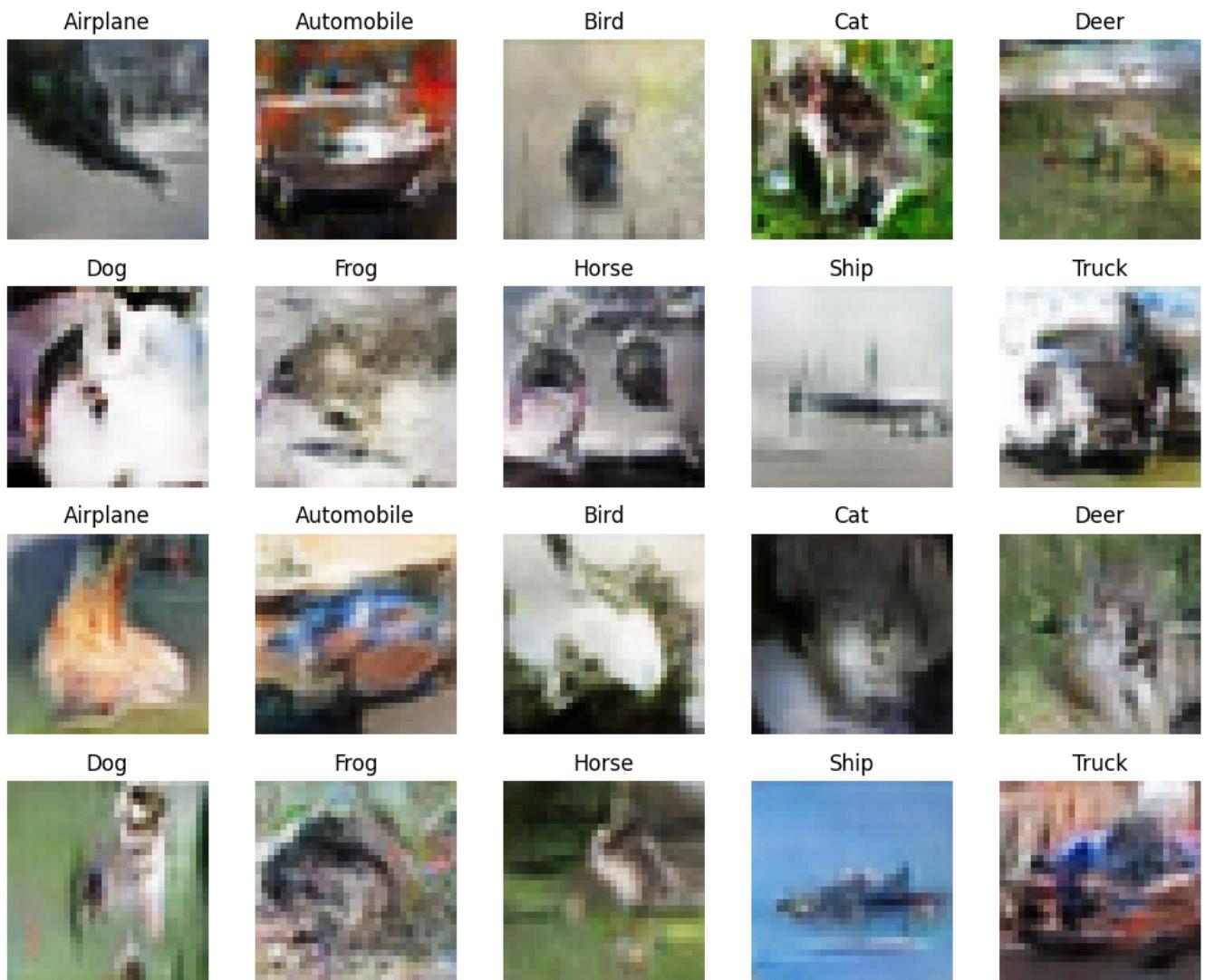
Epoch 44/200

782/782 [=====] - 125s 159ms/step - d_loss: 0.1061 - g_loss: 6.2544 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0550 - KL Divergence: 4.5916

Epoch 45/200

782/782 [=====] - 77s 99ms/step - d_loss: 0.0969 - g_loss: 6.4526 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0523 - KL Divergence: 4.6210

1/1 [=====] - 0s 29ms/step



Generator Checkpoint - cGAN/generator-epoch-45.h5

Epoch 46/200

```
782/782 [=====] - 75s 96ms/step - d_loss: 0.0983 - g_loss: 6.5408 -
D(x|y): 0.5004 - D(G(z|y)): 0.0500 - KL Divergence: 4.5318
```

Epoch 47/200

```
782/782 [=====] - 71s 91ms/step - d_loss: 0.0949 - g_loss: 6.6686 -
D(x|y): 0.5002 - D(G(z|y)): 0.0504 - KL Divergence: 4.6332
```

Epoch 48/200

```
782/782 [=====] - 73s 93ms/step - d_loss: 0.2141 - g_loss: 5.4477 -
D(x|y): 0.5014 - D(G(z|y)): 0.0901 - KL Divergence: 4.7304
```

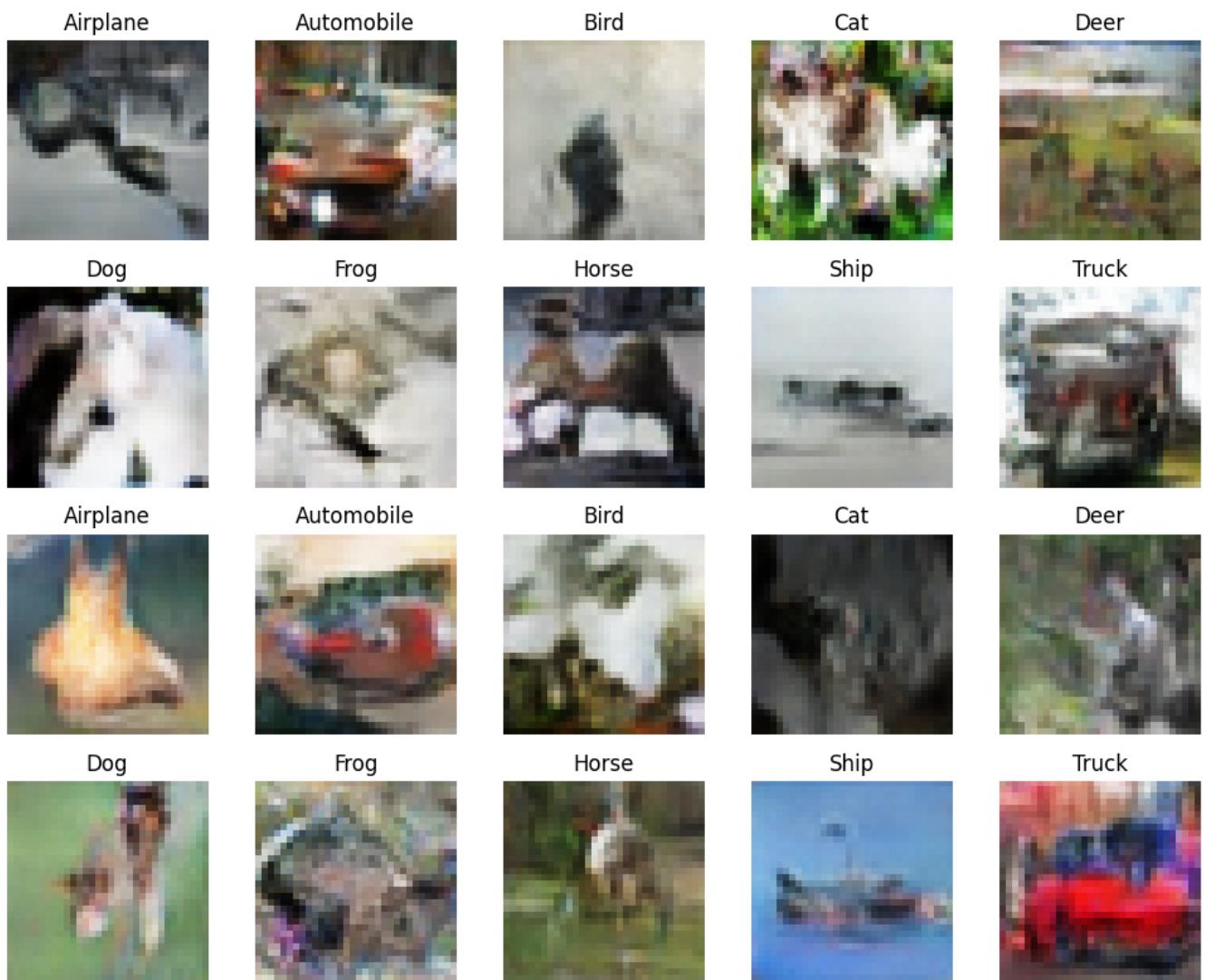
Epoch 49/200

```
782/782 [=====] - 71s 91ms/step - d_loss: 0.0843 - g_loss: 6.7116 -
D(x|y): 0.5000 - D(G(z|y)): 0.0470 - KL Divergence: 4.7106
```

Epoch 50/200

```
782/782 [=====] - 71s 90ms/step - d_loss: 0.0924 - g_loss: 6.7227 -
D(x|y): 0.5001 - D(G(z|y)): 0.0501 - KL Divergence: 4.6189
```

1/1 [=====] - 0s 28ms/step



Generator Checkpoint - cGAN/generator-epoch-50.h5

Epoch 51/200

782/782 [=====] - 70s 90ms/step - d_loss: 0.0903 - g_loss: 6.8311 - D(x|y): 0.5002 - D(G(z|y)): 0.0485 - KL Divergence: 4.5783

Epoch 52/200

782/782 [=====] - 73s 94ms/step - d_loss: 0.0870 - g_loss: 6.9528 - D(x|y): 0.5001 - D(G(z|y)): 0.0465 - KL Divergence: 4.4725

Epoch 53/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0865 - g_loss: 6.9041 - D(x|y): 0.4998 - D(G(z|y)): 0.0481 - KL Divergence: 4.4299

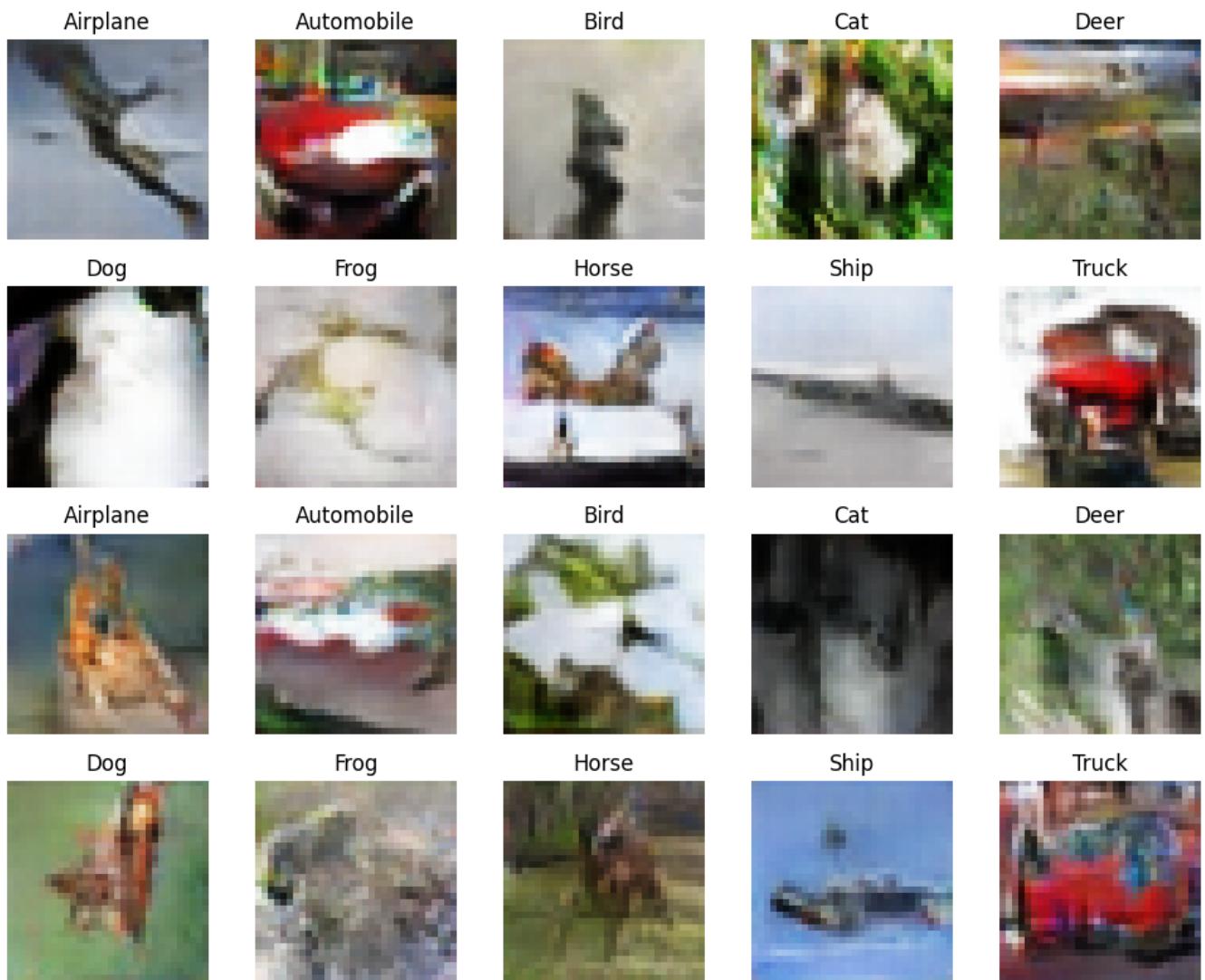
Epoch 54/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0868 - g_loss: 7.0610 - D(x|y): 0.4998 - D(G(z|y)): 0.0438 - KL Divergence: 4.4405

Epoch 55/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0894 - g_loss: 7.0356 - D(x|y): 0.4998 - D(G(z|y)): 0.0496 - KL Divergence: 4.5403

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - cGAN/generator-epoch-55.h5

Epoch 56/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0816 - g_loss: 7.2499 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0415 - KL Divergence: 4.4725

Epoch 57/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0814 - g_loss: 7.3587 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0422 - KL Divergence: 4.4042

Epoch 58/200

782/782 [=====] - 73s 94ms/step - d_loss: 0.0778 - g_loss: 7.3986 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0426 - KL Divergence: 4.5116

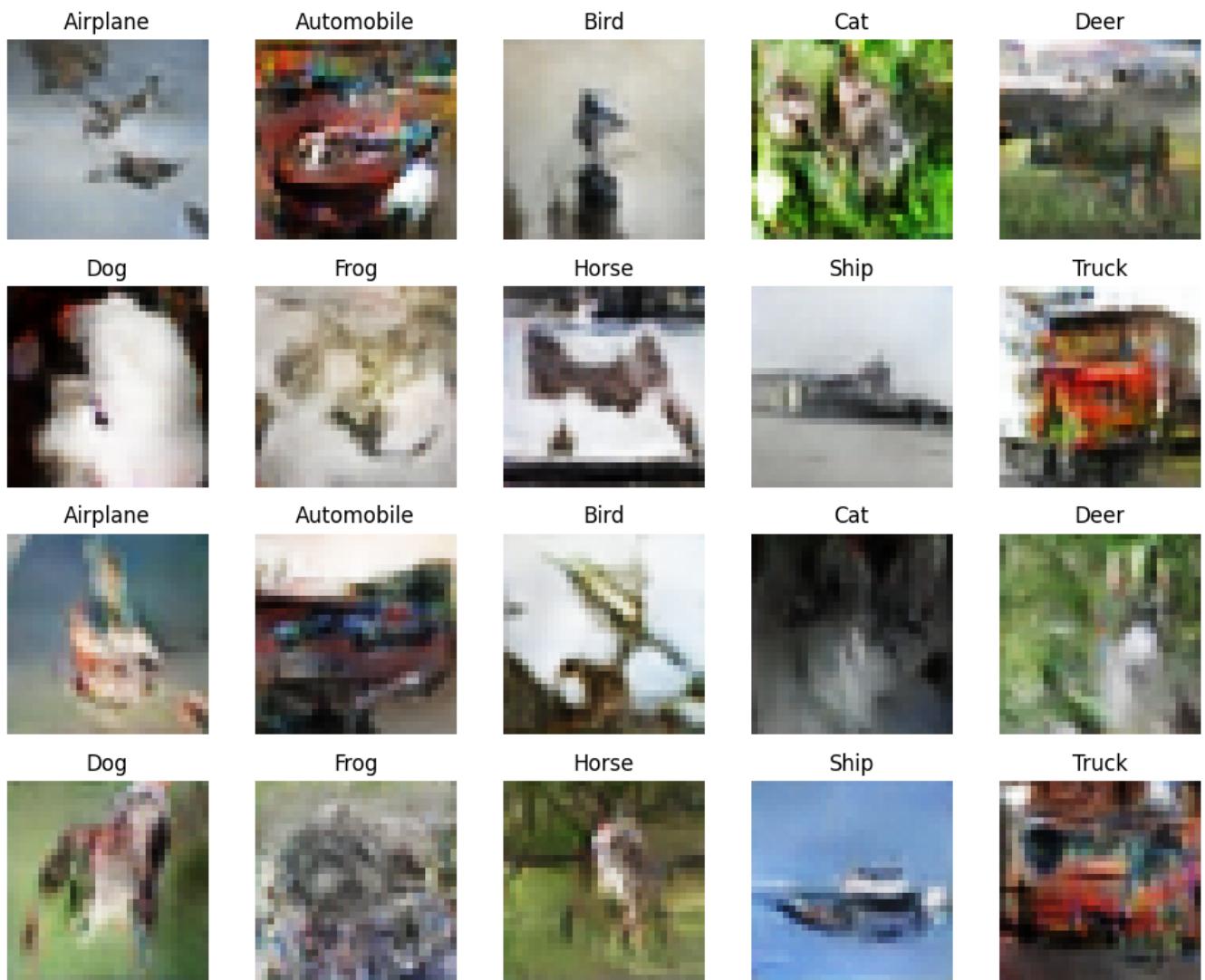
Epoch 59/200

782/782 [=====] - 74s 95ms/step - d_loss: 0.0782 - g_loss: 7.2905 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0435 - KL Divergence: 4.5159

Epoch 60/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0770 - g_loss: 7.4403 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0422 - KL Divergence: 4.5372

1/1 [=====] - 0s 40ms/step



Generator Checkpoint - cGAN/generator-epoch-60.h5

Epoch 61/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0815 - g_loss: 7.5033 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0431 - KL Divergence: 4.3453

Epoch 62/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.0812 - g_loss: 7.5420 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0422 - KL Divergence: 4.4354

Epoch 63/200

782/782 [=====] - 63s 80ms/step - d_loss: 0.0775 - g_loss: 7.6043 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0419 - KL Divergence: 4.4274

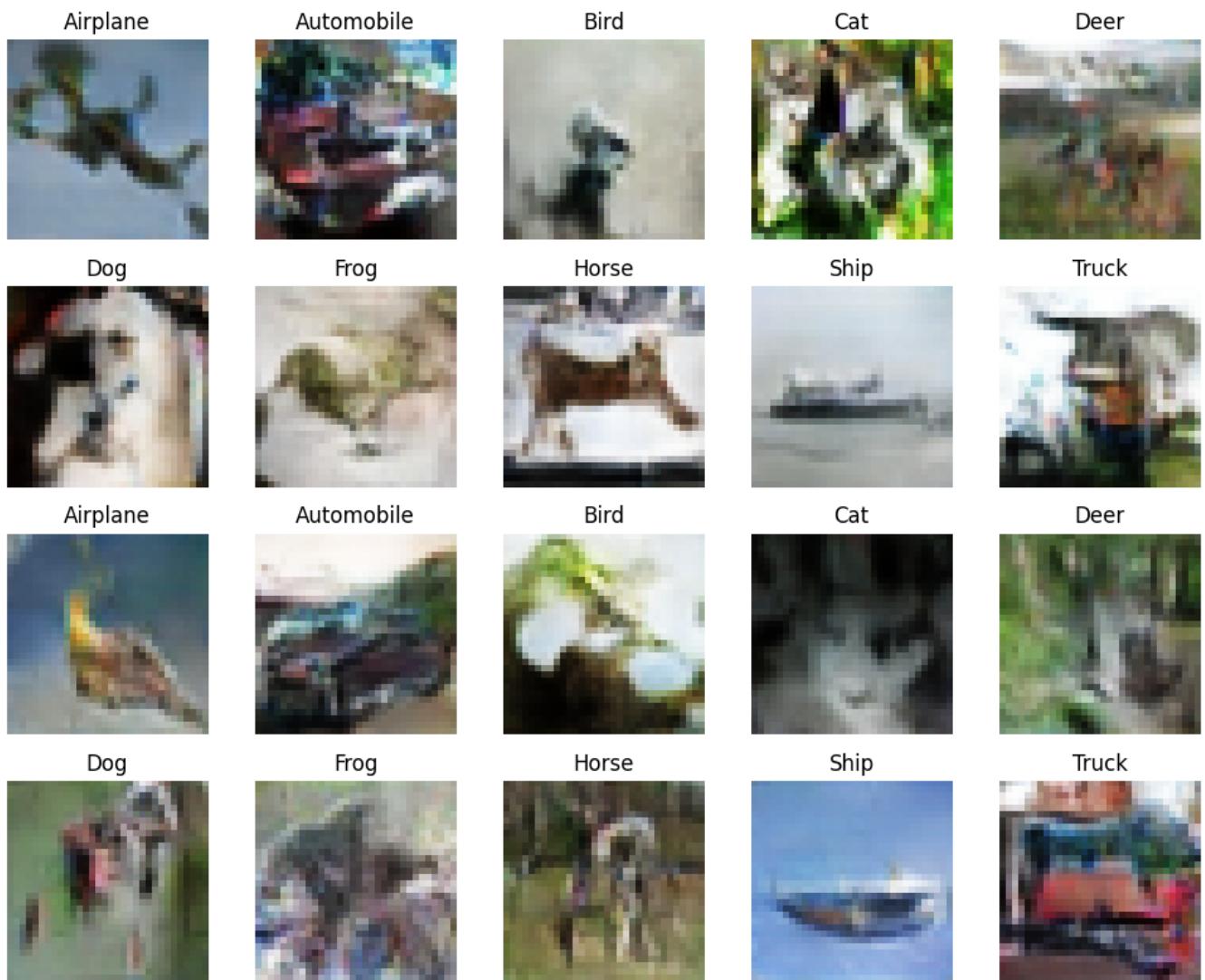
Epoch 64/200

782/782 [=====] - 63s 80ms/step - d_loss: 0.0742 - g_loss: 7.7003 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0423 - KL Divergence: 4.4476

Epoch 65/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0726 - g_loss: 7.8266 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0387 - KL Divergence: 4.3770

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - cGAN/generator-epoch-65.h5

Epoch 66/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0795 - g_loss: 7.7018 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0380$ - KL Divergence: 4.4405

Epoch 67/200

782/782 [=====] - 66s 84ms/step - d_loss: 0.0758 - g_loss: 7.7602 -
 $D(x|y): 0.5005$ - $D(G(z|y)): 0.0407$ - KL Divergence: 4.5301

Epoch 68/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0706 - g_loss: 7.9645 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0373$ - KL Divergence: 4.4222

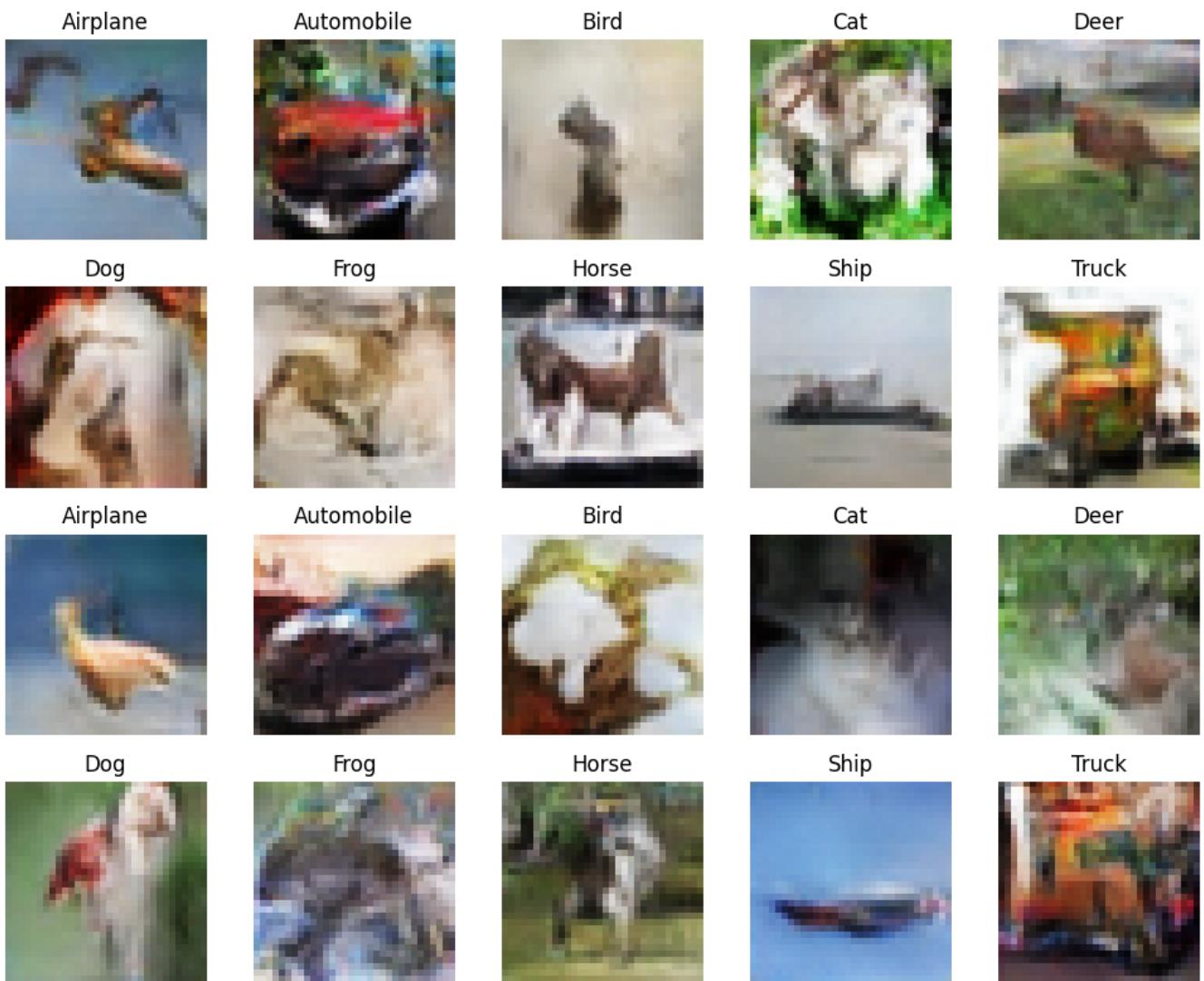
Epoch 69/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0718 - g_loss: 8.1038 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0357$ - KL Divergence: 4.4970

Epoch 70/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0751 - g_loss: 7.9405 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.0370$ - KL Divergence: 4.5157

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - cGAN/generator-epoch-70.h5

Epoch 71/200

782/782 [=====] - 59s 76ms/step - d_loss: 0.0748 - g_loss: 7.9451 - D(x|y): 0.4997 - D(G(z|y)): 0.0393 - KL Divergence: 4.5122

Epoch 72/200

782/782 [=====] - 63s 80ms/step - d_loss: 0.0688 - g_loss: 8.1707 - D(x|y): 0.5003 - D(G(z|y)): 0.0372 - KL Divergence: 4.5427

Epoch 73/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0668 - g_loss: 8.2270 - D(x|y): 0.5001 - D(G(z|y)): 0.0364 - KL Divergence: 4.4146

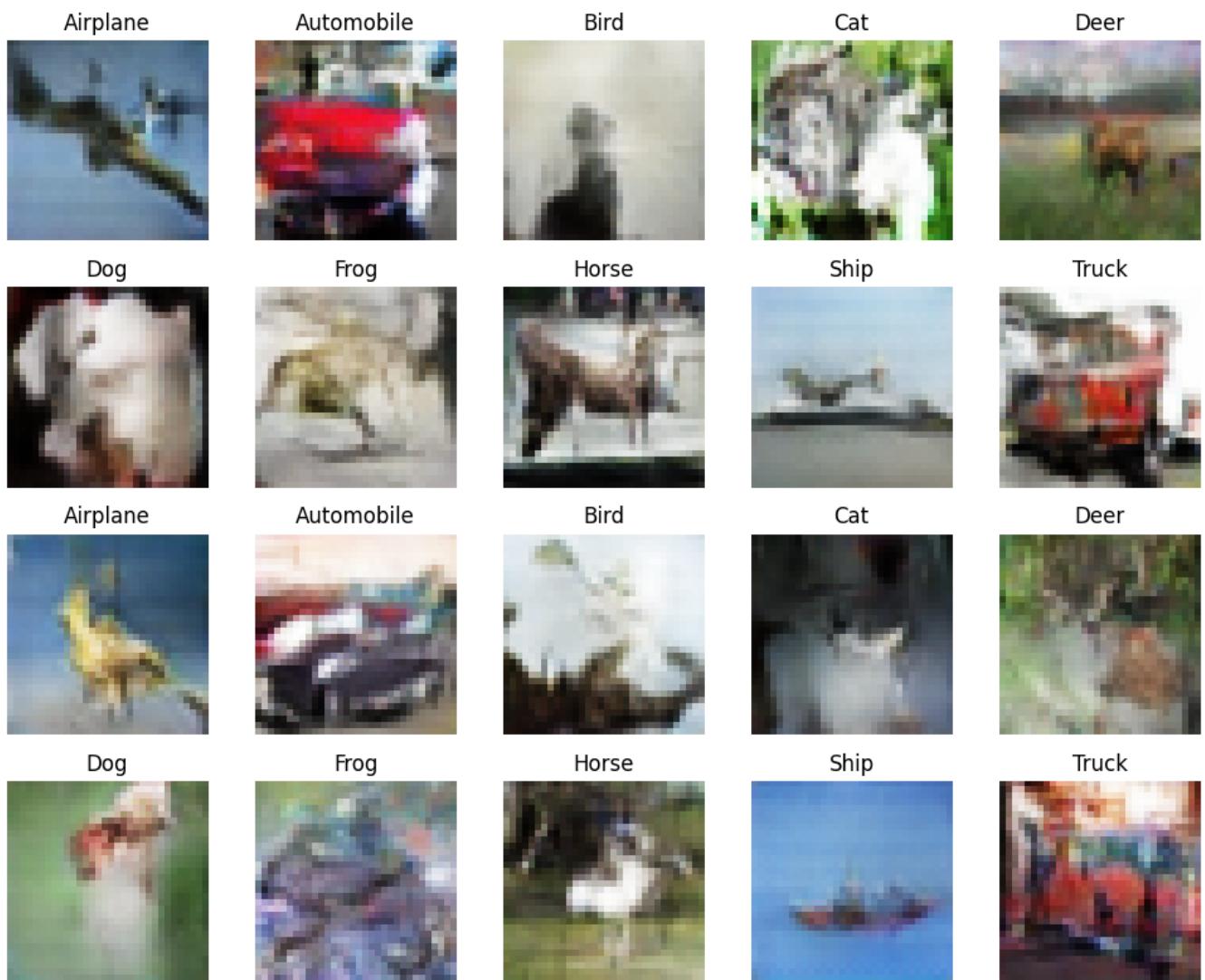
Epoch 74/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0685 - g_loss: 8.1333 - D(x|y): 0.4998 - D(G(z|y)): 0.0350 - KL Divergence: 4.5463

Epoch 75/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0692 - g_loss: 8.0679 - D(x|y): 0.4996 - D(G(z|y)): 0.0370 - KL Divergence: 4.5825

1/1 [=====] - 0s 35ms/step



Generator Checkpoint - cGAN/generator-epoch-75.h5

Epoch 76/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0704 - g_loss: 8.0912 - D(x|y): 0.5001 - D(G(z|y)): 0.0369 - KL Divergence: 4.6126

Epoch 77/200

782/782 [=====] - 63s 80ms/step - d_loss: 0.0652 - g_loss: 8.3150 - D(x|y): 0.5000 - D(G(z|y)): 0.0353 - KL Divergence: 4.6889

Epoch 78/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0660 - g_loss: 8.3525 - D(x|y): 0.5005 - D(G(z|y)): 0.0345 - KL Divergence: 4.4762

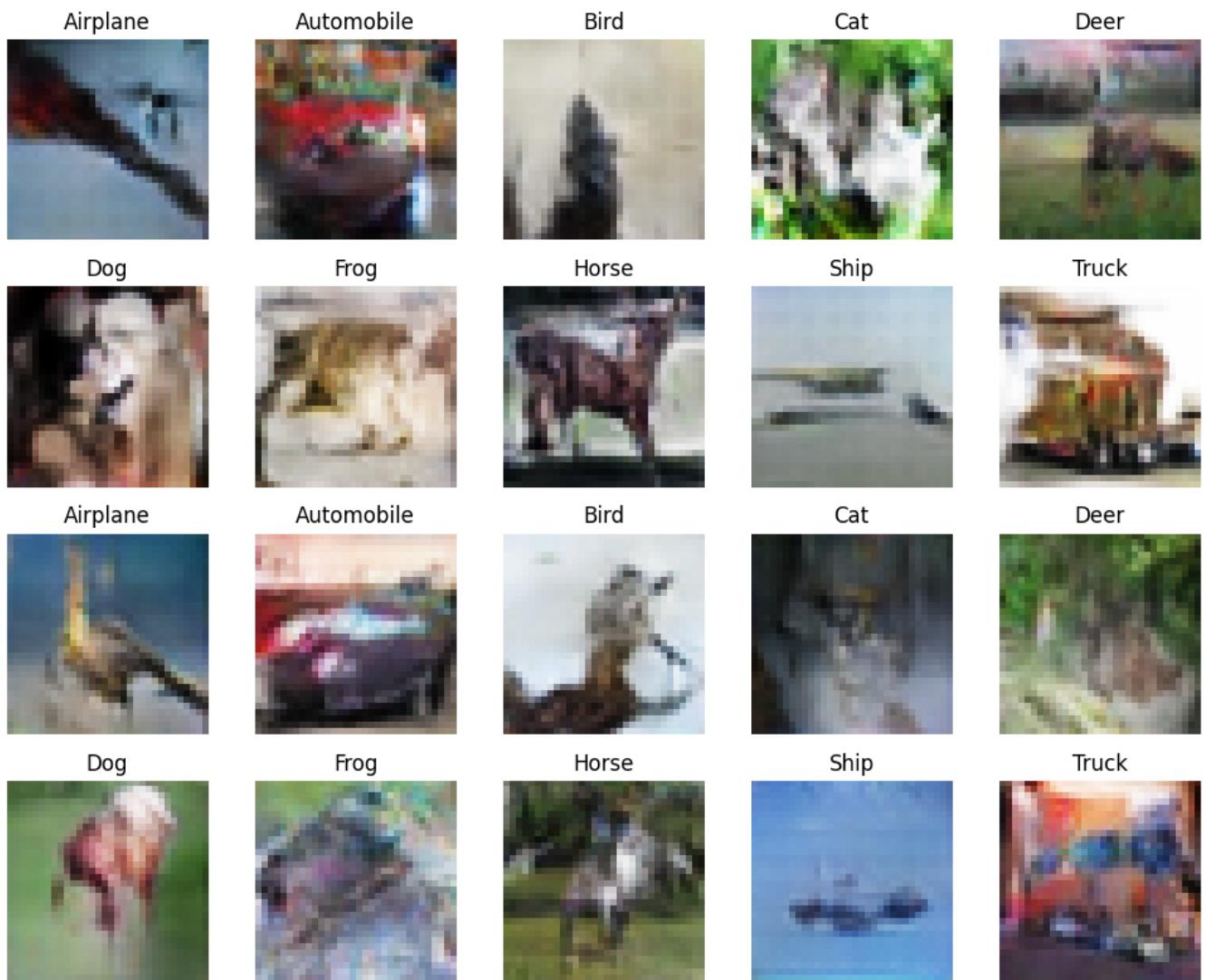
Epoch 79/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0754 - g_loss: 8.2982 - D(x|y): 0.4997 - D(G(z|y)): 0.0380 - KL Divergence: 4.3649

Epoch 80/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0650 - g_loss: 8.3727 - D(x|y): 0.4999 - D(G(z|y)): 0.0355 - KL Divergence: 4.4189

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - cGAN/generator-epoch-80.h5

Epoch 81/200

782/782 [=====] - 65s 84ms/step - d_loss: 0.0635 - g_loss: 8.5000 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0334 - KL Divergence: 4.5453

Epoch 82/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0617 - g_loss: 8.5895 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0337 - KL Divergence: 4.3807

Epoch 83/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0618 - g_loss: 8.5598 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0326 - KL Divergence: 4.6094

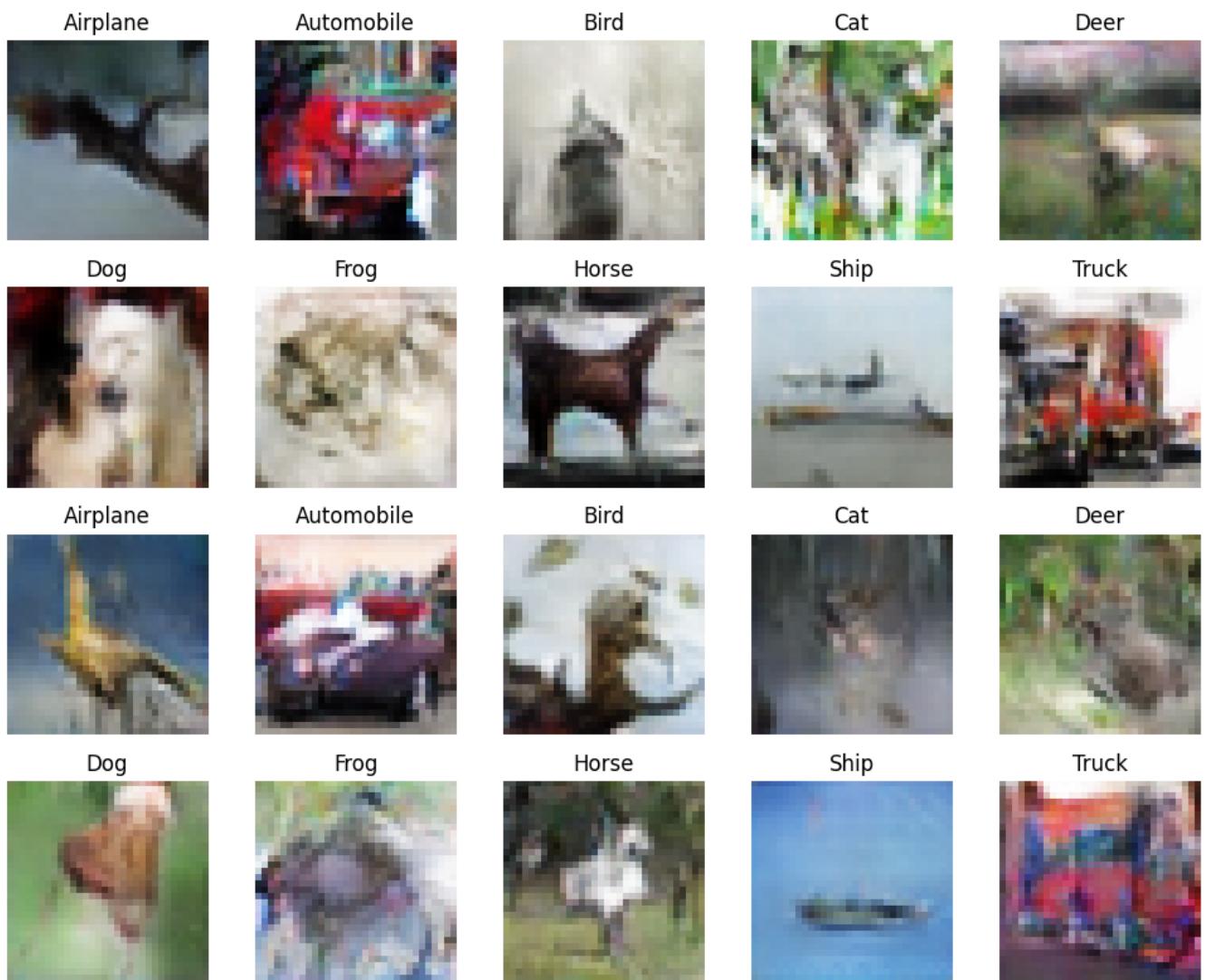
Epoch 84/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0629 - g_loss: 8.6259 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0332 - KL Divergence: 4.5424

Epoch 85/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0632 - g_loss: 8.6963 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0314 - KL Divergence: 4.5816

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - cGAN/generator-epoch-85.h5

Epoch 86/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0584 - g_loss: 8.8067 - D(x|y): 0.5001 - D(G(z|y)): 0.0322 - KL Divergence: 4.4520

Epoch 87/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0642 - g_loss: 8.6613 - D(x|y): 0.5002 - D(G(z|y)): 0.0298 - KL Divergence: 4.4762

Epoch 88/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0601 - g_loss: 8.8067 - D(x|y): 0.5001 - D(G(z|y)): 0.0315 - KL Divergence: 4.4198

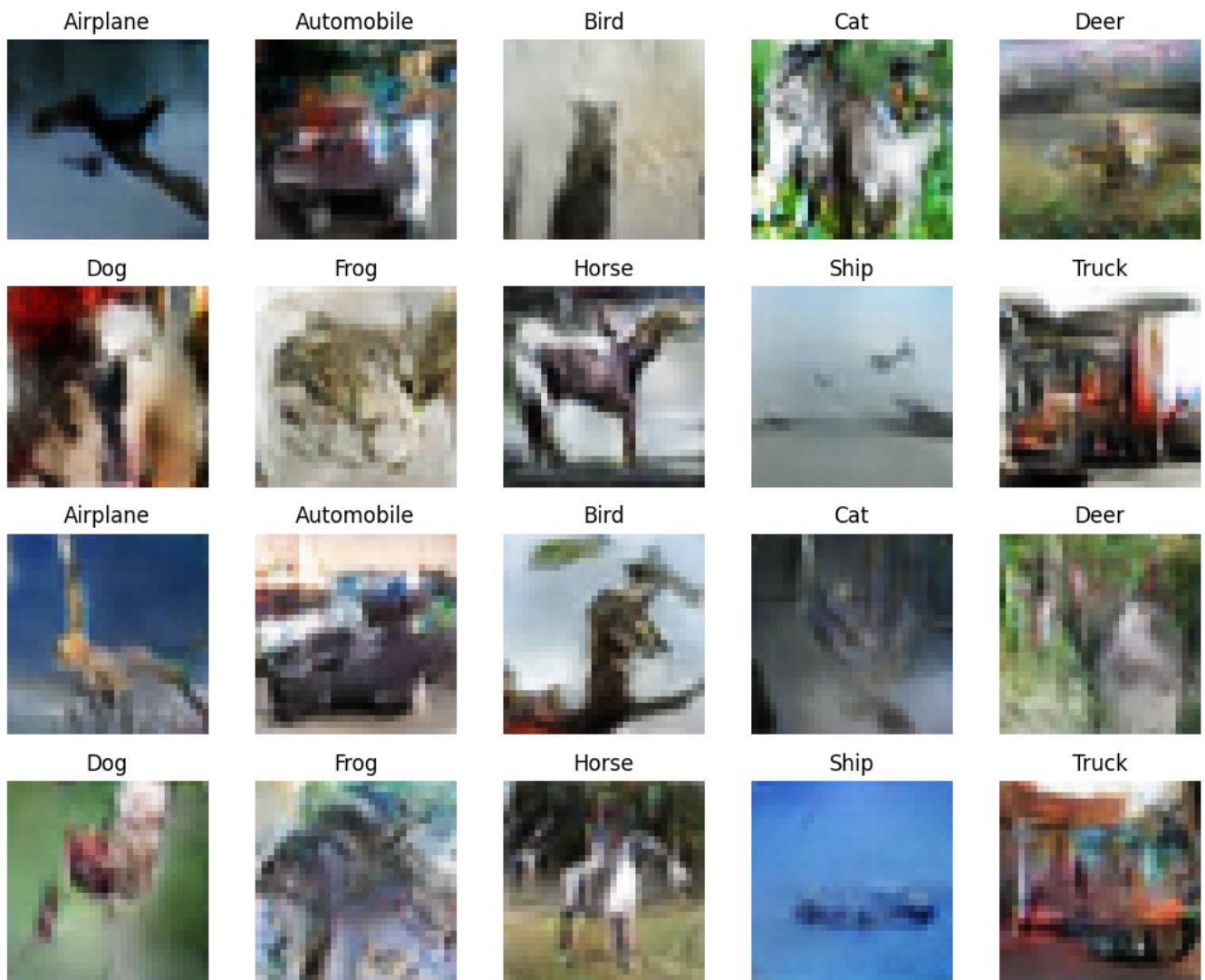
Epoch 89/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0585 - g_loss: 8.7492 - D(x|y): 0.5003 - D(G(z|y)): 0.0309 - KL Divergence: 4.5800

Epoch 90/200

782/782 [=====] - 72s 93ms/step - d_loss: 0.0626 - g_loss: 8.8255 - D(x|y): 0.4996 - D(G(z|y)): 0.0306 - KL Divergence: 4.5677

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - cGAN/generator-epoch-90.h5

Epoch 91/200

782/782 [=====] - 64s 81ms/step - d_loss: 0.0638 - g_loss: 8.7785 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0345 - KL Divergence: 4.6347

Epoch 92/200

782/782 [=====] - 64s 81ms/step - d_loss: 0.0586 - g_loss: 8.8176 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0318 - KL Divergence: 4.5735

Epoch 93/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0558 - g_loss: 8.9601 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0311 - KL Divergence: 4.5888

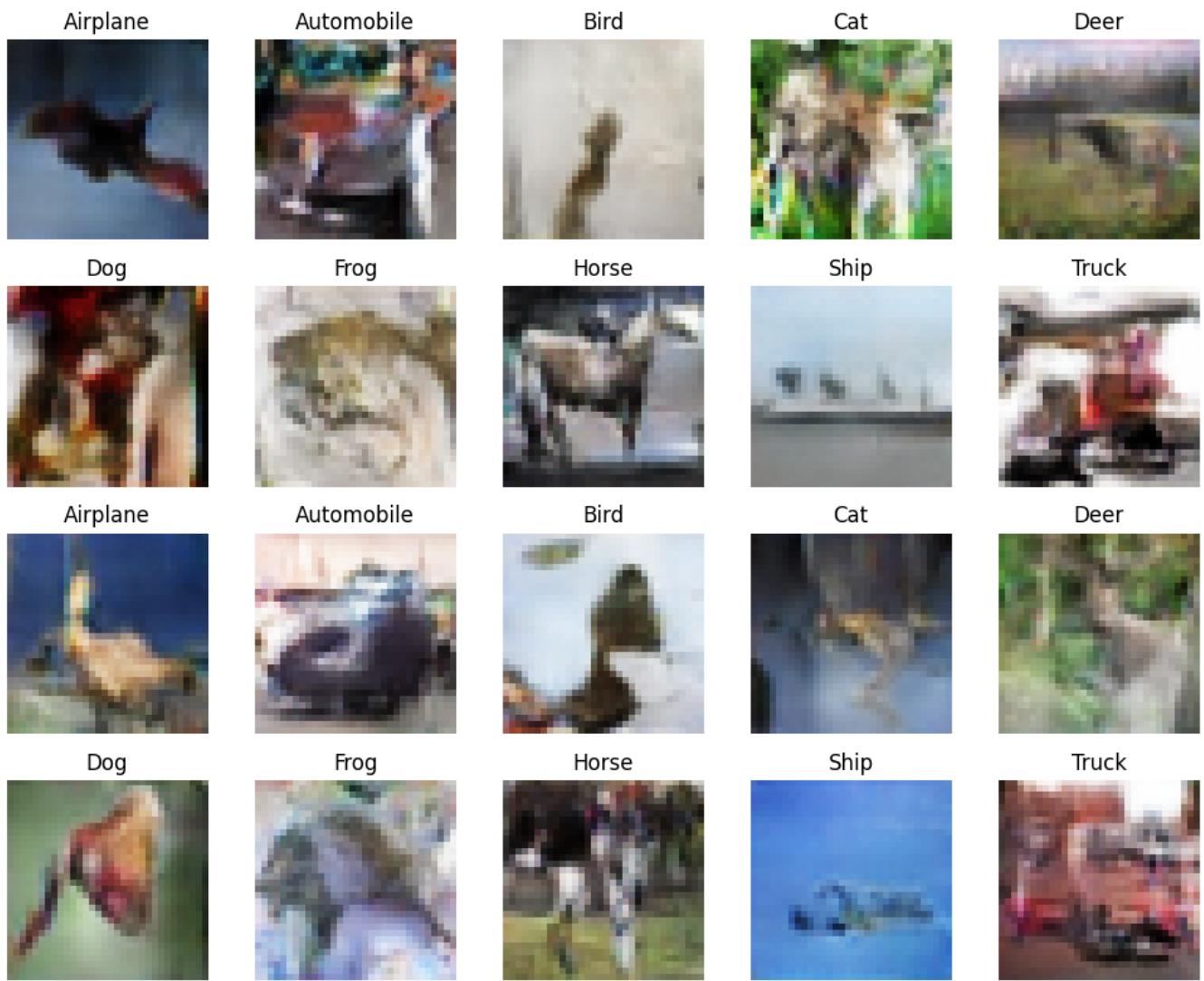
Epoch 94/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0605 - g_loss: 8.9461 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0310 - KL Divergence: 4.5973

Epoch 95/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0575 - g_loss: 9.0080 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0288 - KL Divergence: 4.5335

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - cGAN/generator-epoch-95.h5

Epoch 96/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0579 - g_loss: 8.8230 - D(x|y): 0.5002 - D(G(z|y)): 0.0306 - KL Divergence: 4.5771

Epoch 97/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0537 - g_loss: 9.0884 - D(x|y): 0.5002 - D(G(z|y)): 0.0297 - KL Divergence: 4.6725

Epoch 98/200

782/782 [=====] - 70s 89ms/step - d_loss: 0.0591 - g_loss: 9.0935 - D(x|y): 0.5001 - D(G(z|y)): 0.0303 - KL Divergence: 4.7145

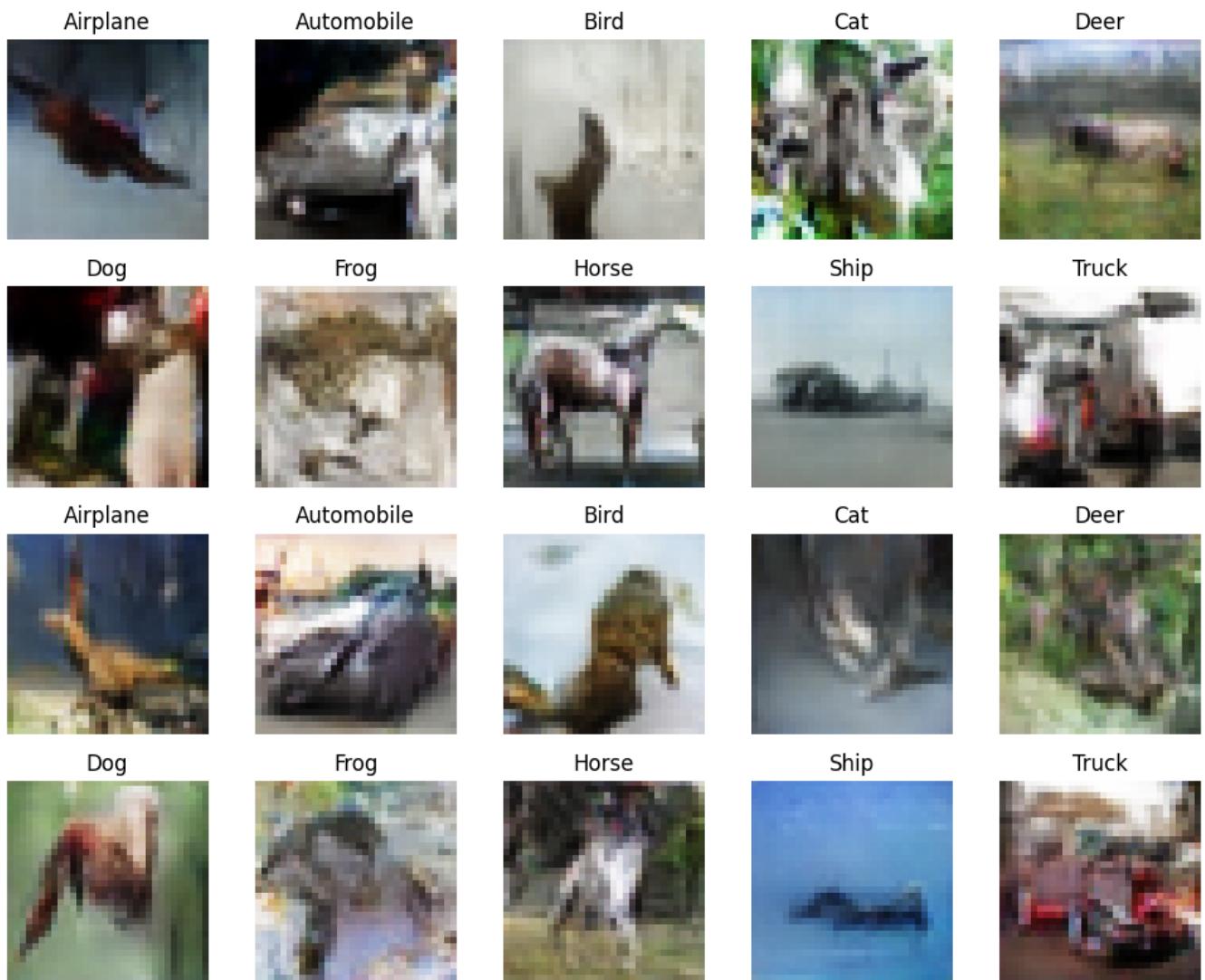
Epoch 99/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.0587 - g_loss: 9.0100 - D(x|y): 0.5000 - D(G(z|y)): 0.0277 - KL Divergence: 4.6664

Epoch 100/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0596 - g_loss: 9.0655 - D(x|y): 0.5000 - D(G(z|y)): 0.0321 - KL Divergence: 4.6086

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - cGAN/generator-epoch-100.h5

Epoch 101/200

782/782 [=====] - 73s 94ms/step - d_loss: 0.0546 - g_loss: 9.2400 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0277 - KL Divergence: 4.4846

Epoch 102/200

782/782 [=====] - 71s 91ms/step - d_loss: 0.0509 - g_loss: 9.3053 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0281 - KL Divergence: 4.6203

Epoch 103/200

782/782 [=====] - 74s 95ms/step - d_loss: 0.0579 - g_loss: 9.1309 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0301 - KL Divergence: 4.5321

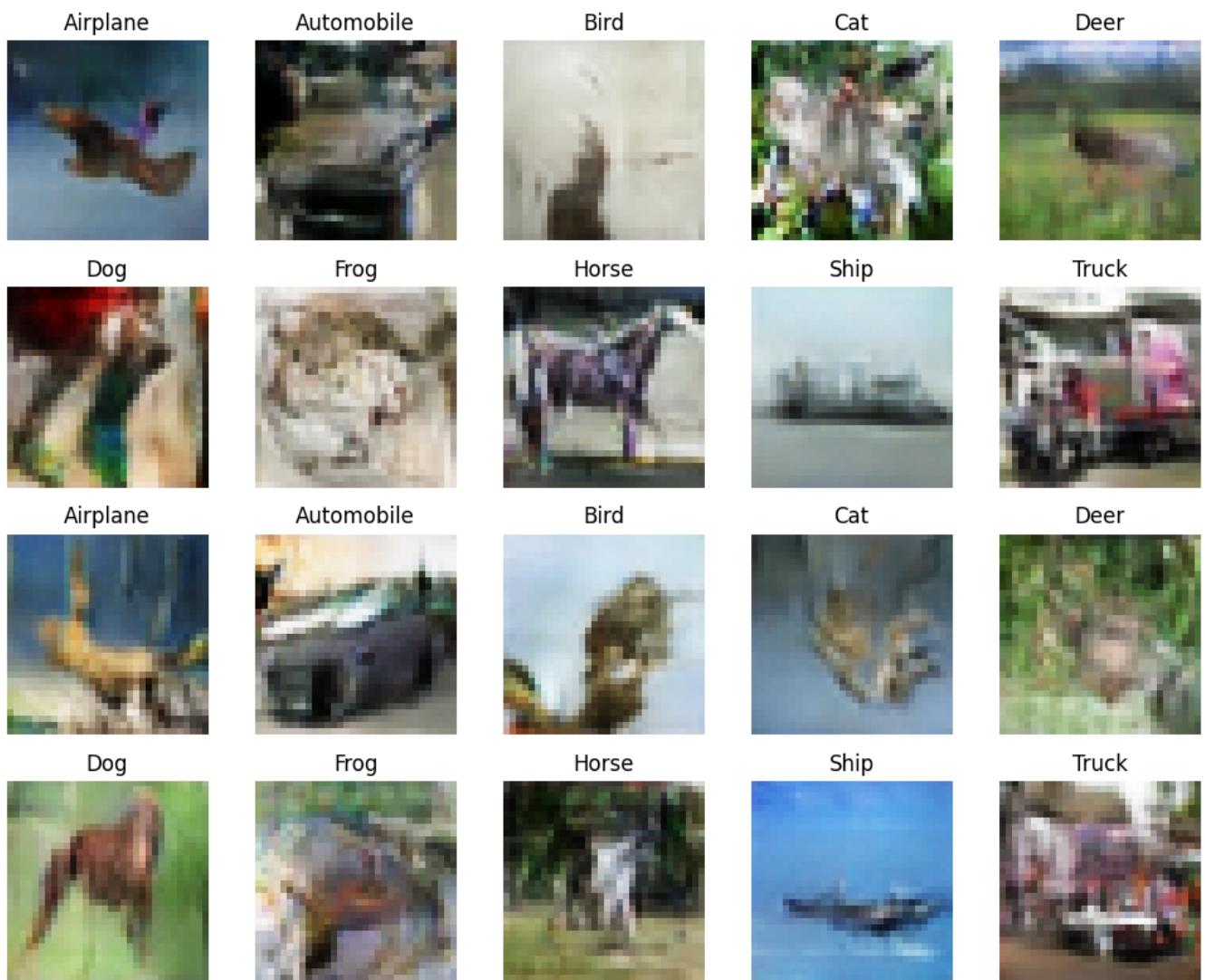
Epoch 104/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.0558 - g_loss: 9.4042 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0285 - KL Divergence: 4.5307

Epoch 105/200

782/782 [=====] - 71s 91ms/step - d_loss: 0.0576 - g_loss: 9.2568 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0301 - KL Divergence: 4.6307

1/1 [=====] - 0s 30ms/step



Generator Checkpoint - cGAN/generator-epoch-105.h5

Epoch 106/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.0539 - g_loss: 9.3884 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0293 - KL Divergence: 4.6269

Epoch 107/200

782/782 [=====] - 70s 90ms/step - d_loss: 0.0525 - g_loss: 9.5311 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0273 - KL Divergence: 4.5779

Epoch 108/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0596 - g_loss: 9.2875 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0299 - KL Divergence: 4.5937

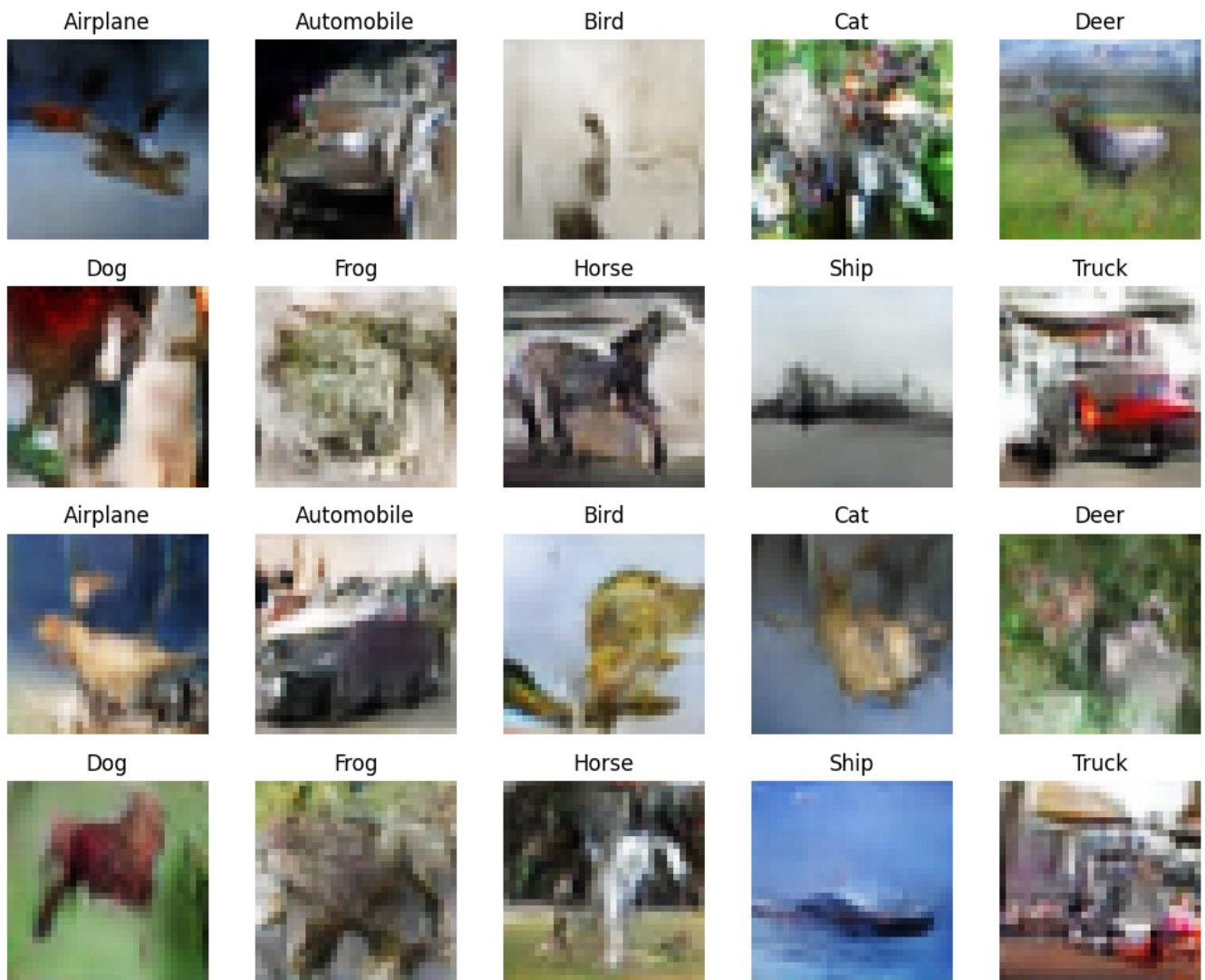
Epoch 109/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0524 - g_loss: 9.4086 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0285 - KL Divergence: 4.6130

Epoch 110/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0496 - g_loss: 9.6614 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0259 - KL Divergence: 4.5116

1/1 [=====] - 0s 22ms/step



Generator Checkpoint - cGAN/generator-epoch-110.h5

Epoch 111/200

782/782 [=====] - 63s 80ms/step - d_loss: 0.0512 - g_loss: 9.4737 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0263 - KL Divergence: 4.5772

Epoch 112/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0535 - g_loss: 9.5654 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0265 - KL Divergence: 4.5738

Epoch 113/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0581 - g_loss: 9.4941 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0285 - KL Divergence: 4.4917

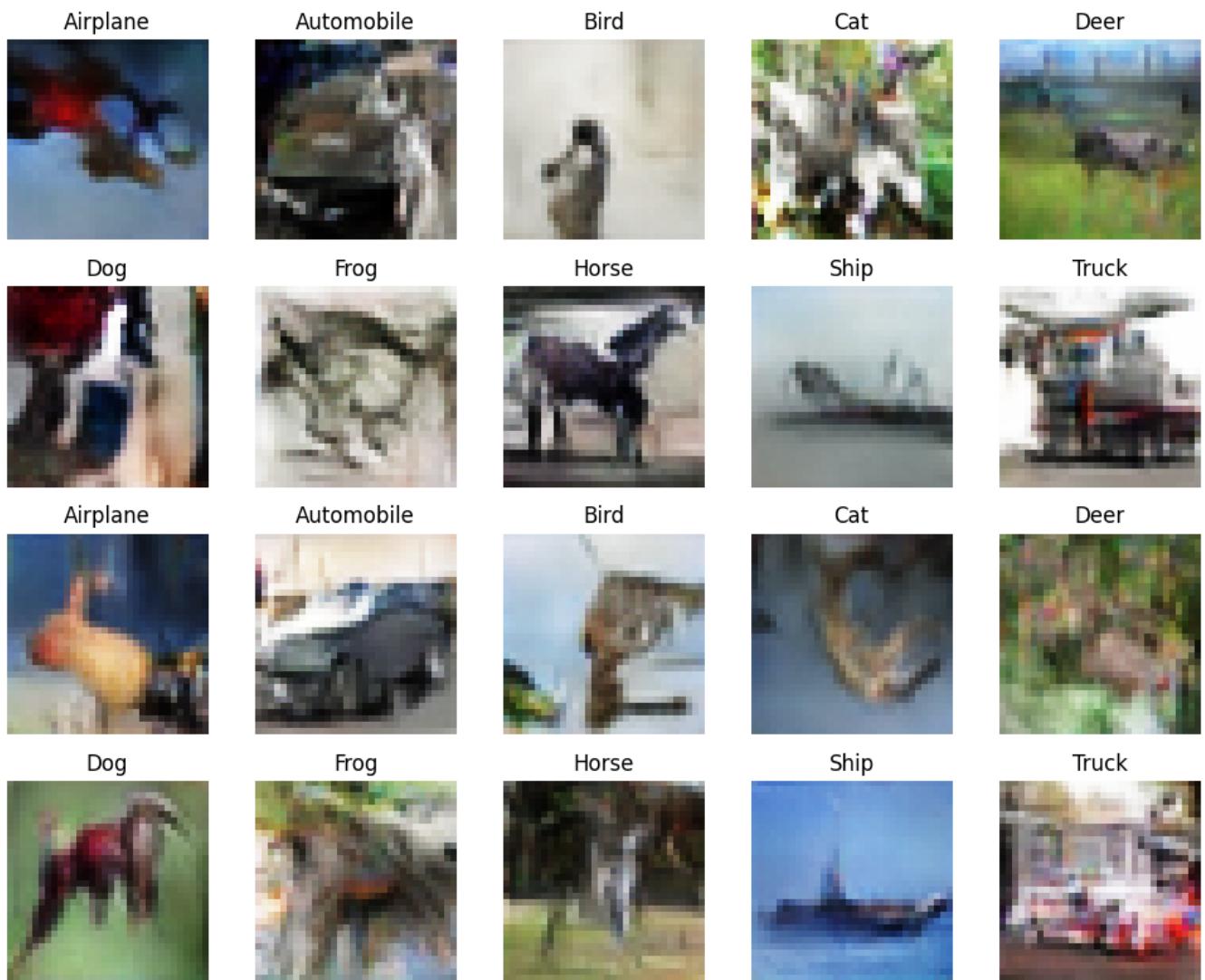
Epoch 114/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0500 - g_loss: 9.7375 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0255 - KL Divergence: 4.5181

Epoch 115/200

782/782 [=====] - 66s 84ms/step - d_loss: 0.0523 - g_loss: 9.6659 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0268 - KL Divergence: 4.4897

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - cGAN/generator-epoch-115.h5

Epoch 116/200

782/782 [=====] - 69s 89ms/step - d_loss: 0.0515 - g_loss: 9.6950 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.0268$ - KL Divergence: 4.4341

Epoch 117/200

782/782 [=====] - 70s 89ms/step - d_loss: 0.0538 - g_loss: 9.6219 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0250$ - KL Divergence: 4.3640

Epoch 118/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0542 - g_loss: 9.5436 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0267$ - KL Divergence: 4.4896

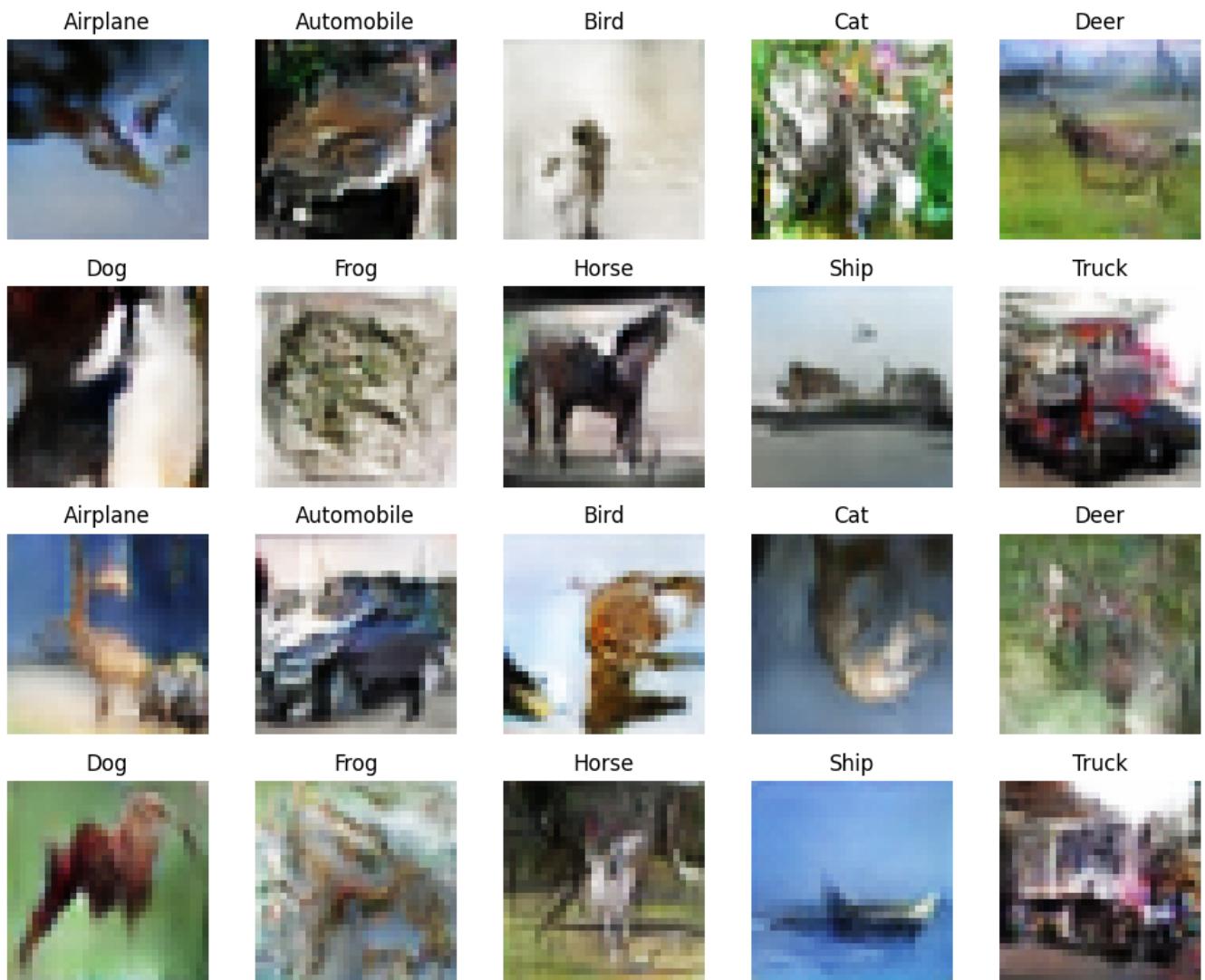
Epoch 119/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.0480 - g_loss: 9.7372 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0265$ - KL Divergence: 4.4639

Epoch 120/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0479 - g_loss: 10.0041 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0249$ - KL Divergence: 4.3933

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - cGAN/generator-epoch-120.h5

Epoch 121/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0490 - g_loss: 9.6872 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0256 - KL Divergence: 4.4661

Epoch 122/200

782/782 [=====] - 66s 85ms/step - d_loss: 0.0512 - g_loss: 9.5724 -
 $D(x|y)$: 0.4997 - $D(G(z|y))$: 0.0272 - KL Divergence: 4.5358

Epoch 123/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0474 - g_loss: 9.8894 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0237 - KL Divergence: 4.6033

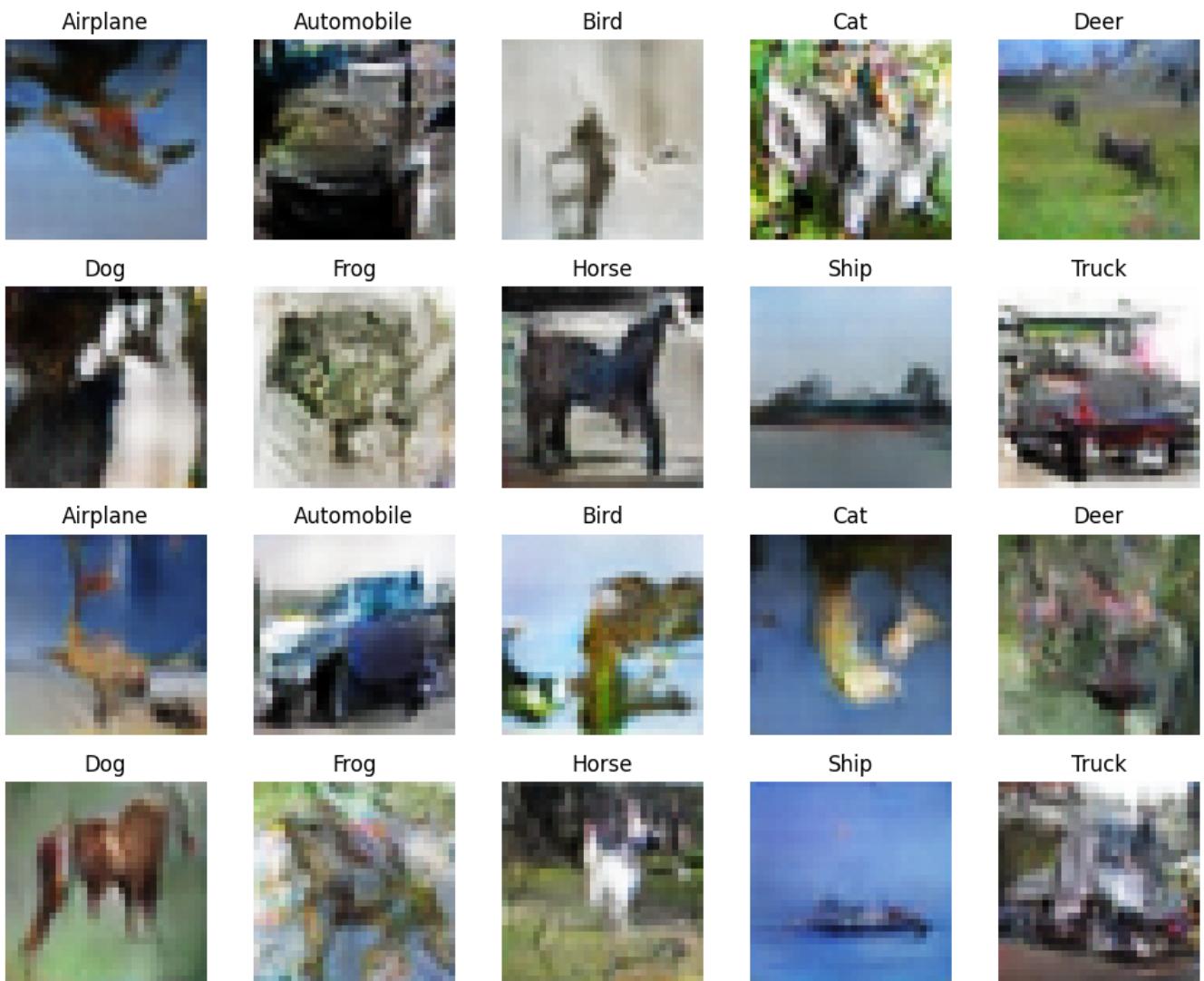
Epoch 124/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0615 - g_loss: 9.4532 -
 $D(x|y)$: 0.4997 - $D(G(z|y))$: 0.0309 - KL Divergence: 4.4342

Epoch 125/200

782/782 [=====] - 73s 94ms/step - d_loss: 0.0454 - g_loss: 10.1005 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0229 - KL Divergence: 4.4813

1/1 [=====] - 0s 81ms/step



Generator Checkpoint - cGAN/generator-epoch-125.h5

Epoch 126/200

782/782 [=====] - 75s 95ms/step - d_loss: 0.0433 - g_loss: 9.9961 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0234 - KL Divergence: 4.6299

Epoch 127/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0496 - g_loss: 9.9761 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0228 - KL Divergence: 4.4763

Epoch 128/200

782/782 [=====] - 75s 96ms/step - d_loss: 0.0467 - g_loss: 9.9507 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0252 - KL Divergence: 4.4770

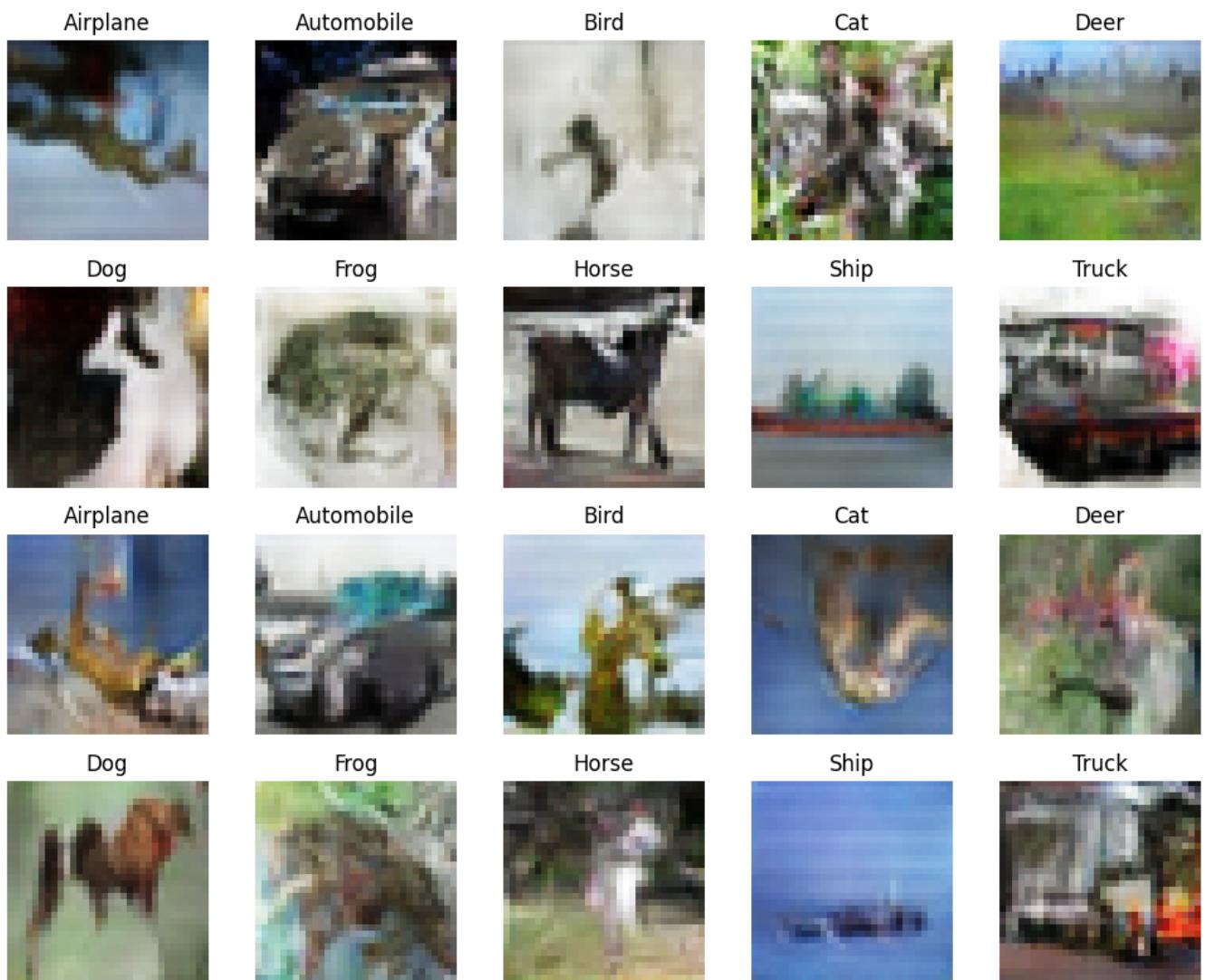
Epoch 129/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.0493 - g_loss: 9.9549 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0244 - KL Divergence: 4.4190

Epoch 130/200

782/782 [=====] - 70s 89ms/step - d_loss: 0.0454 - g_loss: 10.0584 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0228 - KL Divergence: 4.4198

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - cGAN/generator-epoch-130.h5

Epoch 131/200

782/782 [=====] - 71s 91ms/step - d_loss: 0.0494 - g_loss: 9.9569 - D(x|y): 0.5004 - D(G(z|y)): 0.0245 - KL Divergence: 4.3665

Epoch 132/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.0478 - g_loss: 9.8345 - D(x|y): 0.5003 - D(G(z|y)): 0.0248 - KL Divergence: 4.4539

Epoch 133/200

782/782 [=====] - 71s 91ms/step - d_loss: 0.0487 - g_loss: 9.8149 - D(x|y): 0.5000 - D(G(z|y)): 0.0255 - KL Divergence: 4.5078

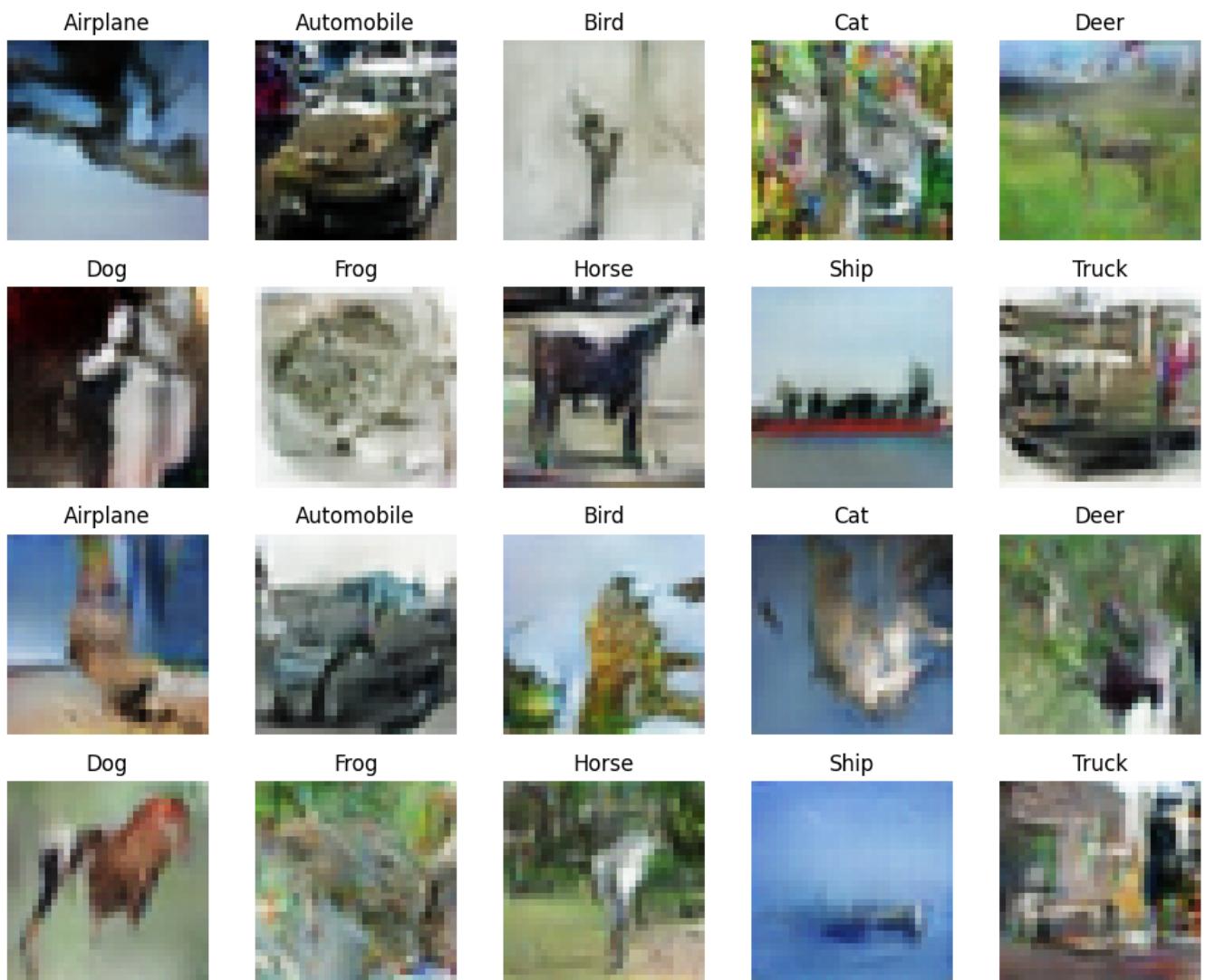
Epoch 134/200

782/782 [=====] - 88s 112ms/step - d_loss: 0.0432 - g_loss: 10.2140 - D(x|y): 0.5000 - D(G(z|y)): 0.0228 - KL Divergence: 4.4664

Epoch 135/200

782/782 [=====] - 109s 140ms/step - d_loss: 0.0471 - g_loss: 10.1809 - D(x|y): 0.5003 - D(G(z|y)): 0.0228 - KL Divergence: 4.5349

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - cGAN/generator-epoch-135.h5

Epoch 136/200

782/782 [=====] - 91s 116ms/step - d_loss: 0.0536 - g_loss: 9.9969 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0266 - KL Divergence: 4.5650

Epoch 137/200

782/782 [=====] - 97s 124ms/step - d_loss: 0.0422 - g_loss: 10.0192
 $- D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0221 - KL Divergence: 4.5449

Epoch 138/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.0488 - g_loss: 9.9873 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0231 - KL Divergence: 4.4575

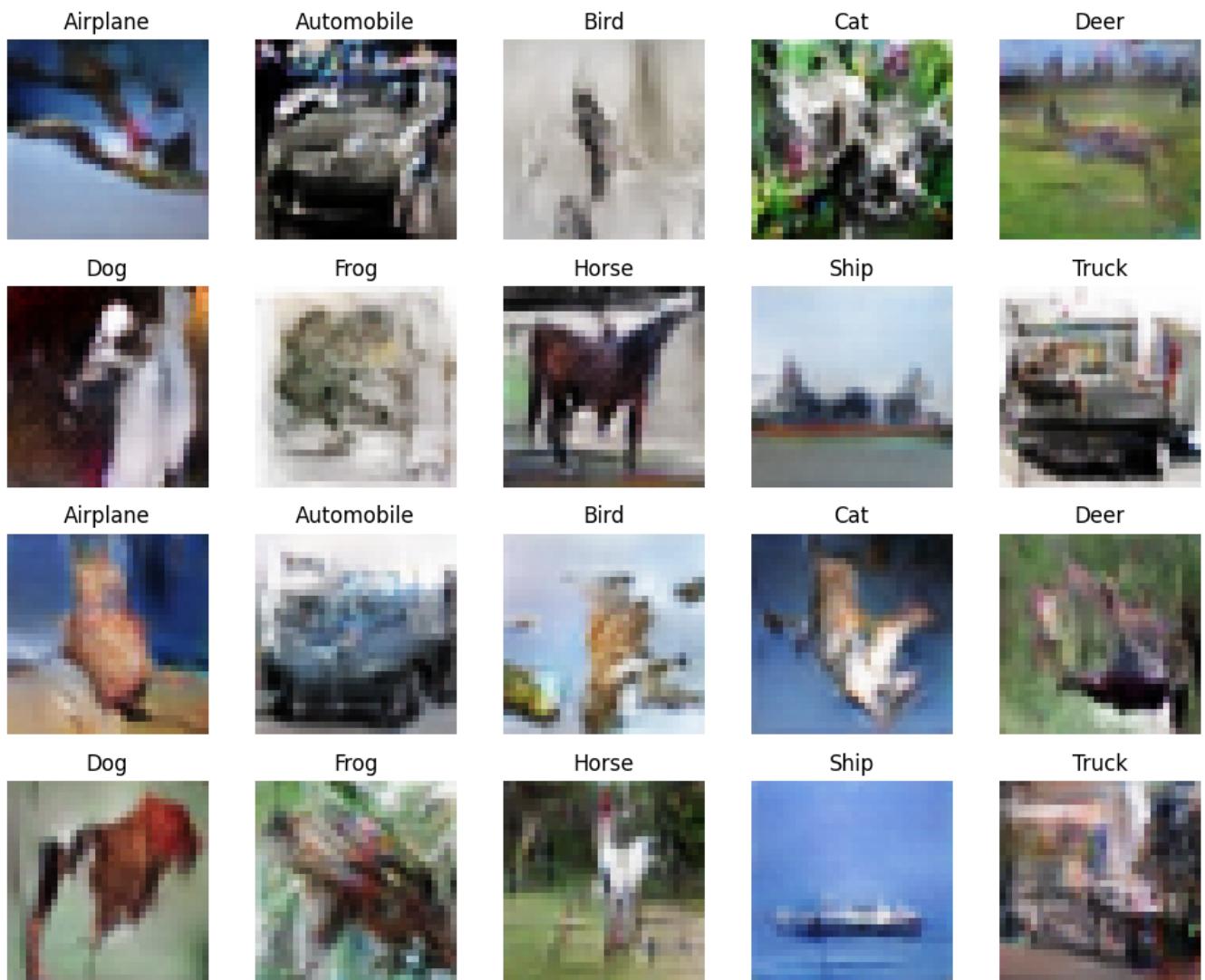
Epoch 139/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.0406 - g_loss: 10.1816
 $- D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0207 - KL Divergence: 4.4748

Epoch 140/200

782/782 [=====] - 93s 119ms/step - d_loss: 0.0412 - g_loss: 10.1282
 $- D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0245 - KL Divergence: 4.3233

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - cGAN/generator-epoch-140.h5

Epoch 141/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.0440 - g_loss: 10.2198
 $- D(x|y): 0.5002 - D(G(z|y)): 0.0218 - KL Divergence: 4.5105$

Epoch 142/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0449 - g_loss: 10.2572 -
 $D(x|y): 0.5001 - D(G(z|y)): 0.0215 - KL Divergence: 4.4829$

Epoch 143/200

782/782 [=====] - 45s 57ms/step - d_loss: 0.0396 - g_loss: 10.2141 -
 $D(x|y): 0.5000 - D(G(z|y)): 0.0222 - KL Divergence: 4.3987$

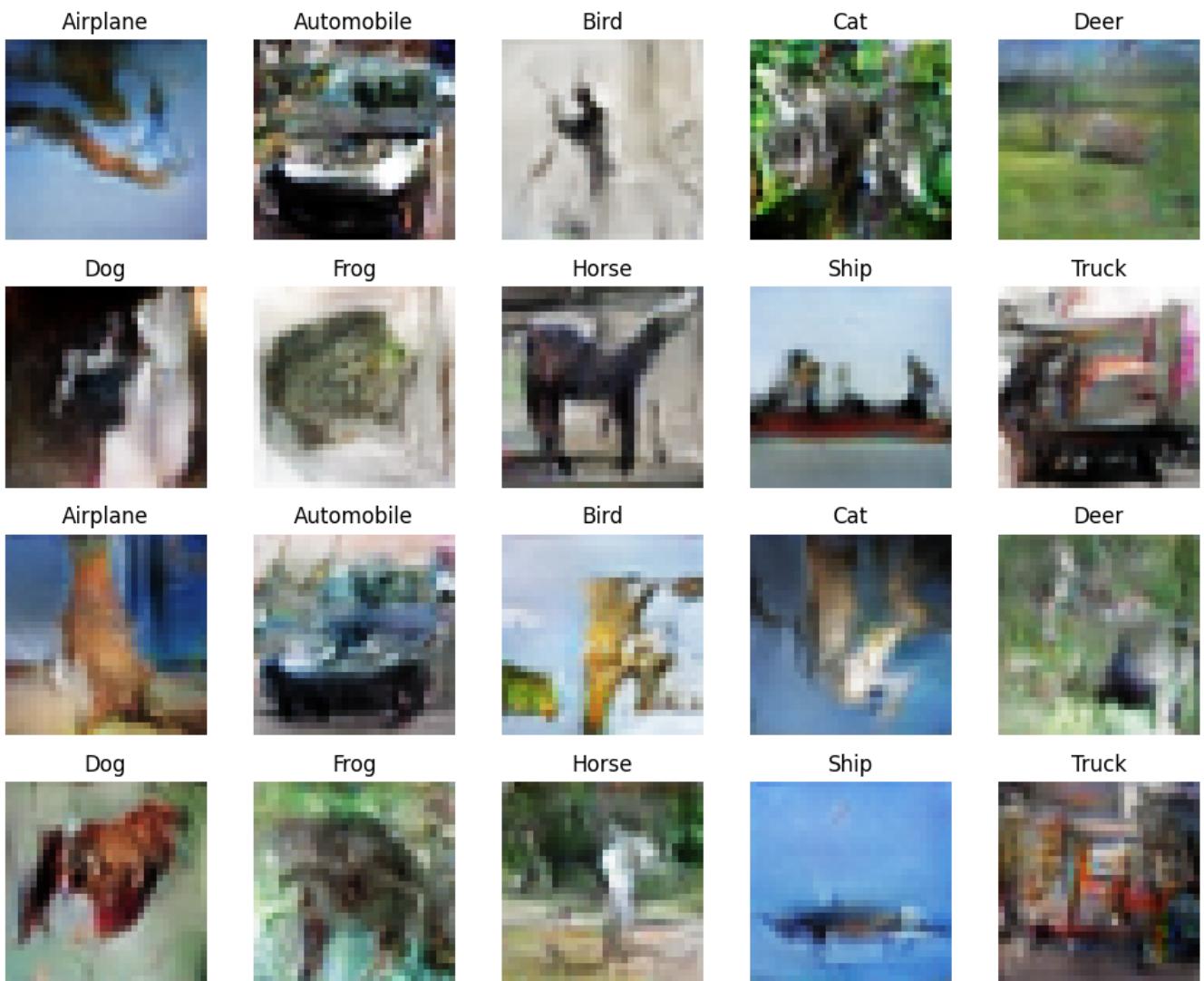
Epoch 144/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0434 - g_loss: 10.3029 -
 $D(x|y): 0.5004 - D(G(z|y)): 0.0222 - KL Divergence: 4.5692$

Epoch 145/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0450 - g_loss: 10.2961 -
 $D(x|y): 0.4997 - D(G(z|y)): 0.0230 - KL Divergence: 4.3560$

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - cGAN/generator-epoch-145.h5

Epoch 146/200

782/782 [=====] - 50s 64ms/step - d_loss: 0.0439 - g_loss: 10.3285 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0213 - KL Divergence: 4.3877

Epoch 147/200

782/782 [=====] - 50s 64ms/step - d_loss: 0.0399 - g_loss: 10.2543 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0219 - KL Divergence: 4.5931

Epoch 148/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0463 - g_loss: 10.0871 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0223 - KL Divergence: 4.5000

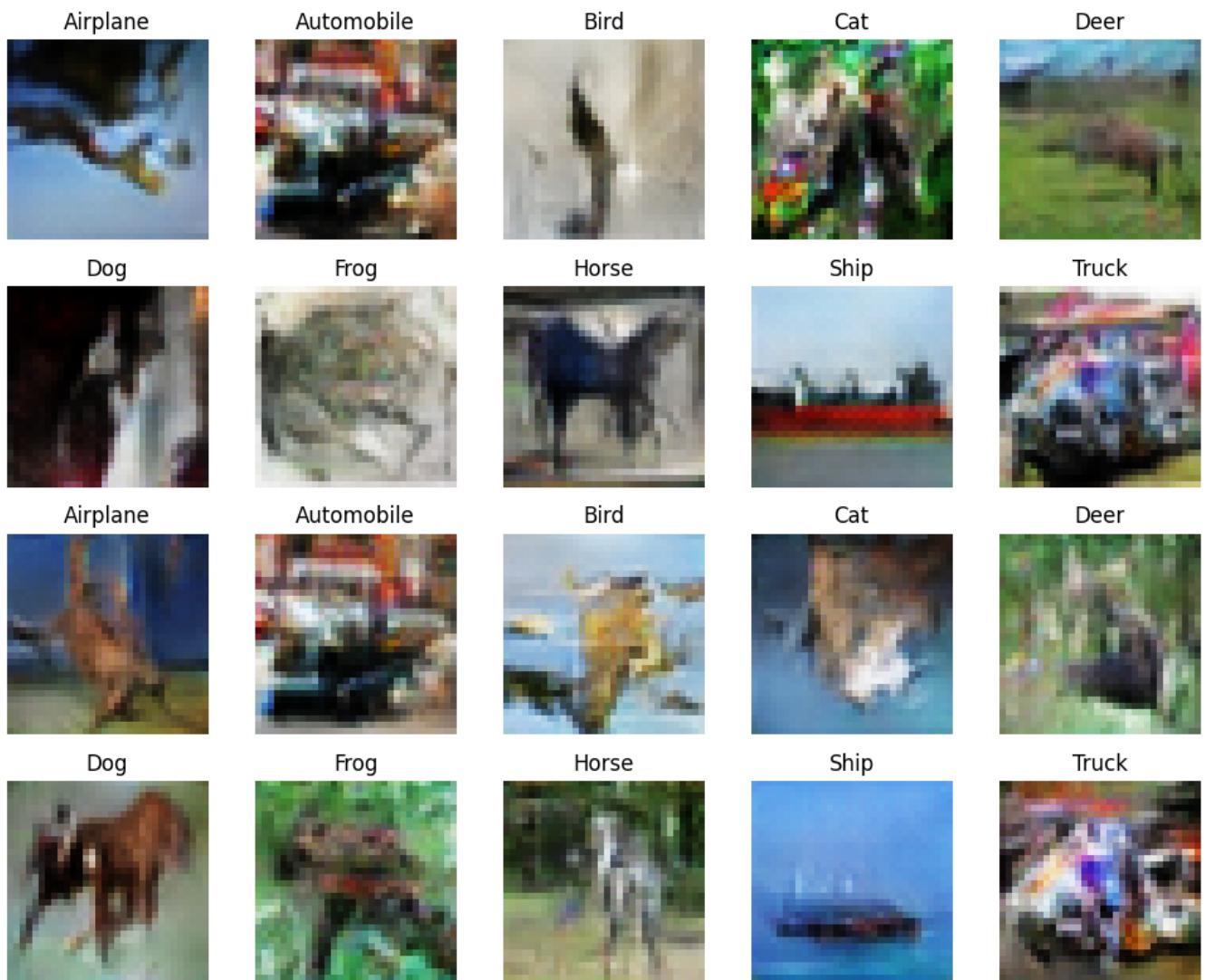
Epoch 149/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0450 - g_loss: 10.0692 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0210 - KL Divergence: 4.5824

Epoch 150/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0429 - g_loss: 10.1244 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0209 - KL Divergence: 4.4747

1/1 [=====] - 0s 31ms/step



Generator Checkpoint - cGAN/generator-epoch-150.h5

Epoch 151/200

782/782 [=====] - 50s 64ms/step - d_loss: 0.0424 - g_loss: 10.0620 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0207 - KL Divergence: 5.0981

Epoch 152/200

782/782 [=====] - 53s 68ms/step - d_loss: 0.0480 - g_loss: 9.8623 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0220 - KL Divergence: 4.7471

Epoch 153/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0478 - g_loss: 9.7459 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0228 - KL Divergence: 4.8388

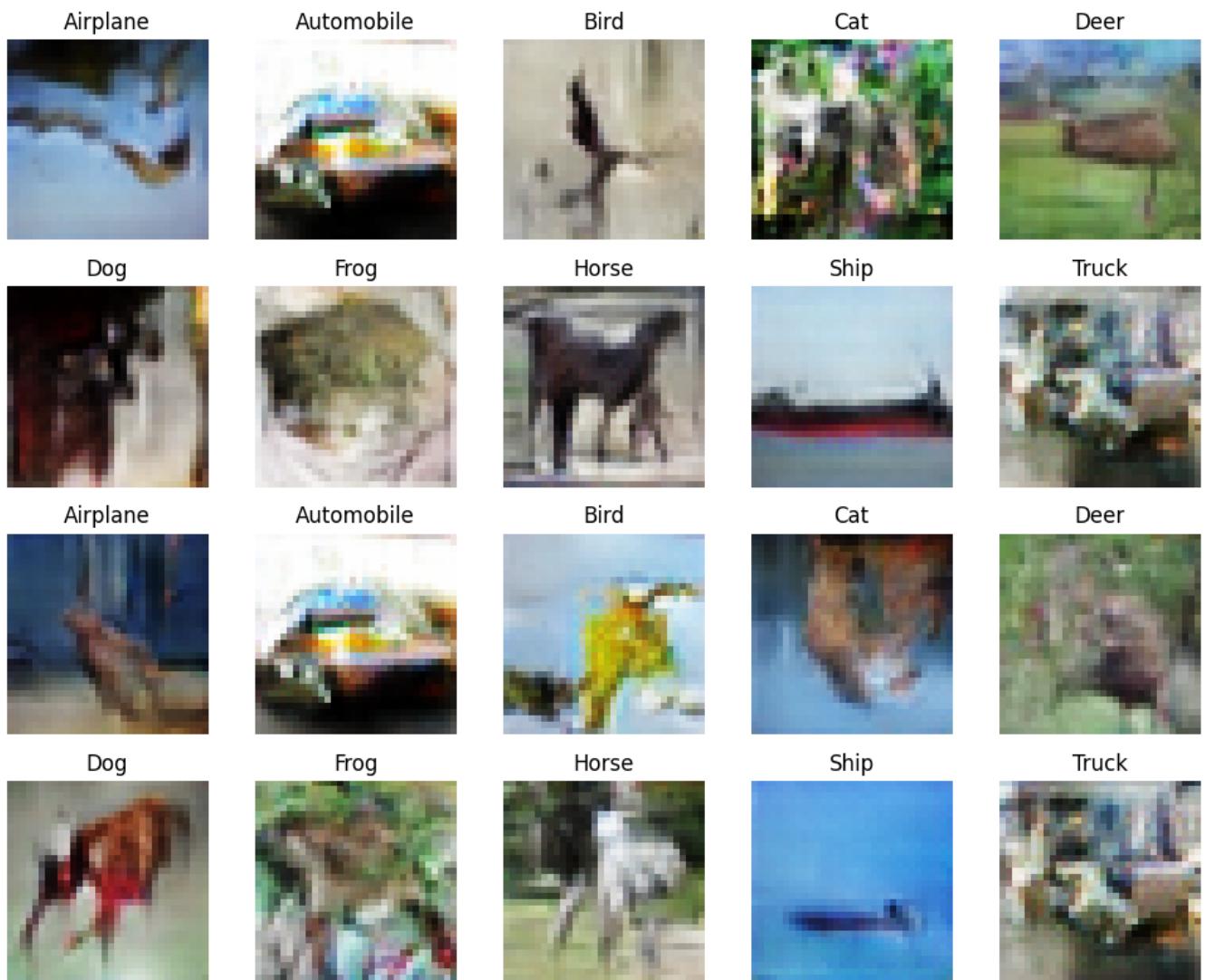
Epoch 154/200

782/782 [=====] - 80s 103ms/step - d_loss: 0.0473 - g_loss: 9.6613 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0213 - KL Divergence: 4.7579

Epoch 155/200

782/782 [=====] - 79s 101ms/step - d_loss: 0.0494 - g_loss: 9.6006 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0223 - KL Divergence: 4.4742

1/1 [=====] - 0s 36ms/step



Generator Checkpoint - cGAN/generator-epoch-155.h5

Epoch 156/200

782/782 [=====] - 75s 96ms/step - d_loss: 0.0507 - g_loss: 9.4706 - D(x|y): 0.5001 - D(G(z|y)): 0.0226 - KL Divergence: 4.4549

Epoch 157/200

782/782 [=====] - 78s 99ms/step - d_loss: 0.0516 - g_loss: 9.5134 - D(x|y): 0.5002 - D(G(z|y)): 0.0234 - KL Divergence: 4.9313

Epoch 158/200

782/782 [=====] - 89s 114ms/step - d_loss: 0.0515 - g_loss: 9.5337 - D(x|y): 0.5005 - D(G(z|y)): 0.0237 - KL Divergence: 4.3586

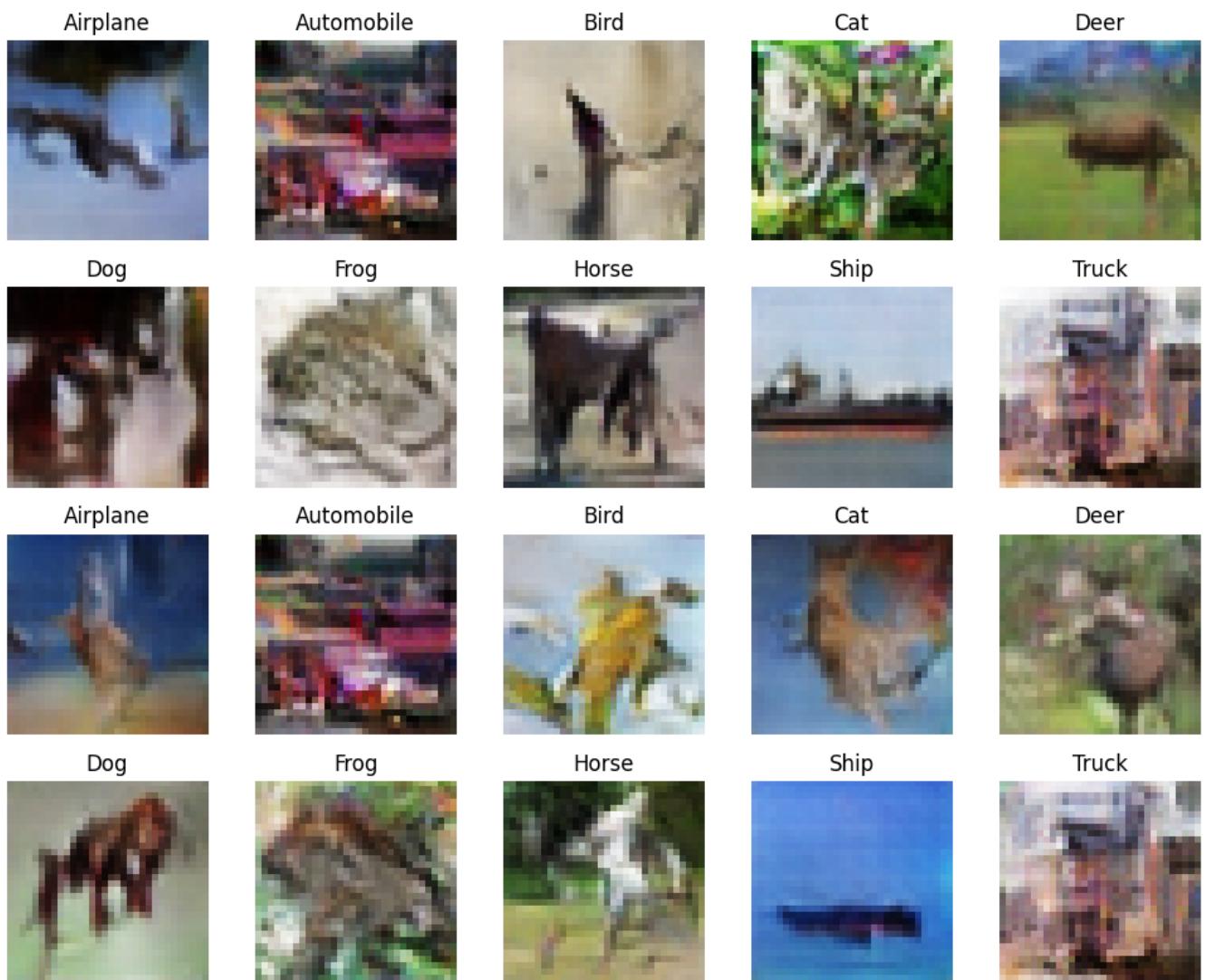
Epoch 159/200

782/782 [=====] - 93s 119ms/step - d_loss: 0.0576 - g_loss: 9.4633 - D(x|y): 0.5001 - D(G(z|y)): 0.0243 - KL Divergence: 4.4471

Epoch 160/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0534 - g_loss: 9.3525 - D(x|y): 0.5007 - D(G(z|y)): 0.0256 - KL Divergence: 4.2428

1/1 [=====] - 0s 29ms/step



Generator Checkpoint - cGAN/generator-epoch-160.h5

Epoch 161/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0553 - g_loss: 9.4093 - D(x|y): 0.5002 - D(G(z|y)): 0.0246 - KL Divergence: 4.6710

Epoch 162/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0495 - g_loss: 9.4292 - D(x|y): 0.5003 - D(G(z|y)): 0.0230 - KL Divergence: 4.6330

Epoch 163/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0573 - g_loss: 9.5676 - D(x|y): 0.5003 - D(G(z|y)): 0.0242 - KL Divergence: 4.6990

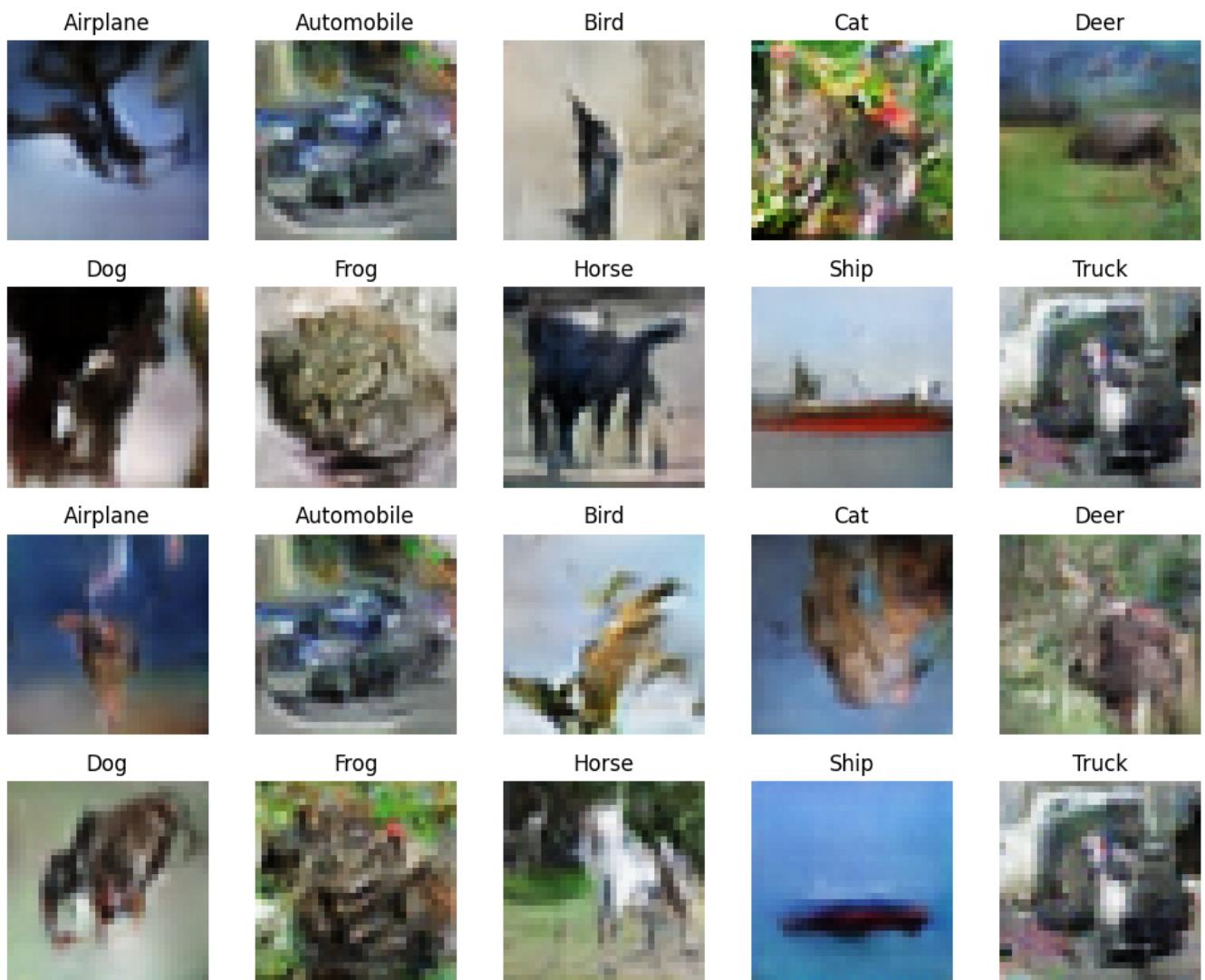
Epoch 164/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0536 - g_loss: 9.3422 - D(x|y): 0.5005 - D(G(z|y)): 0.0239 - KL Divergence: 4.7132

Epoch 165/200

782/782 [=====] - 61s 78ms/step - d_loss: 0.0580 - g_loss: 9.3585 - D(x|y): 0.4996 - D(G(z|y)): 0.0257 - KL Divergence: 4.4489

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - cGAN/generator-epoch-165.h5

Epoch 166/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0587 - g_loss: 9.3344 - D(x|y): 0.5001 - D(G(z|y)): 0.0234 - KL Divergence: 4.6203

Epoch 167/200

782/782 [=====] - 66s 85ms/step - d_loss: 0.0522 - g_loss: 9.3847 - D(x|y): 0.5005 - D(G(z|y)): 0.0229 - KL Divergence: 4.2342

Epoch 168/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.0553 - g_loss: 9.4722 - D(x|y): 0.5001 - D(G(z|y)): 0.0244 - KL Divergence: 4.6620

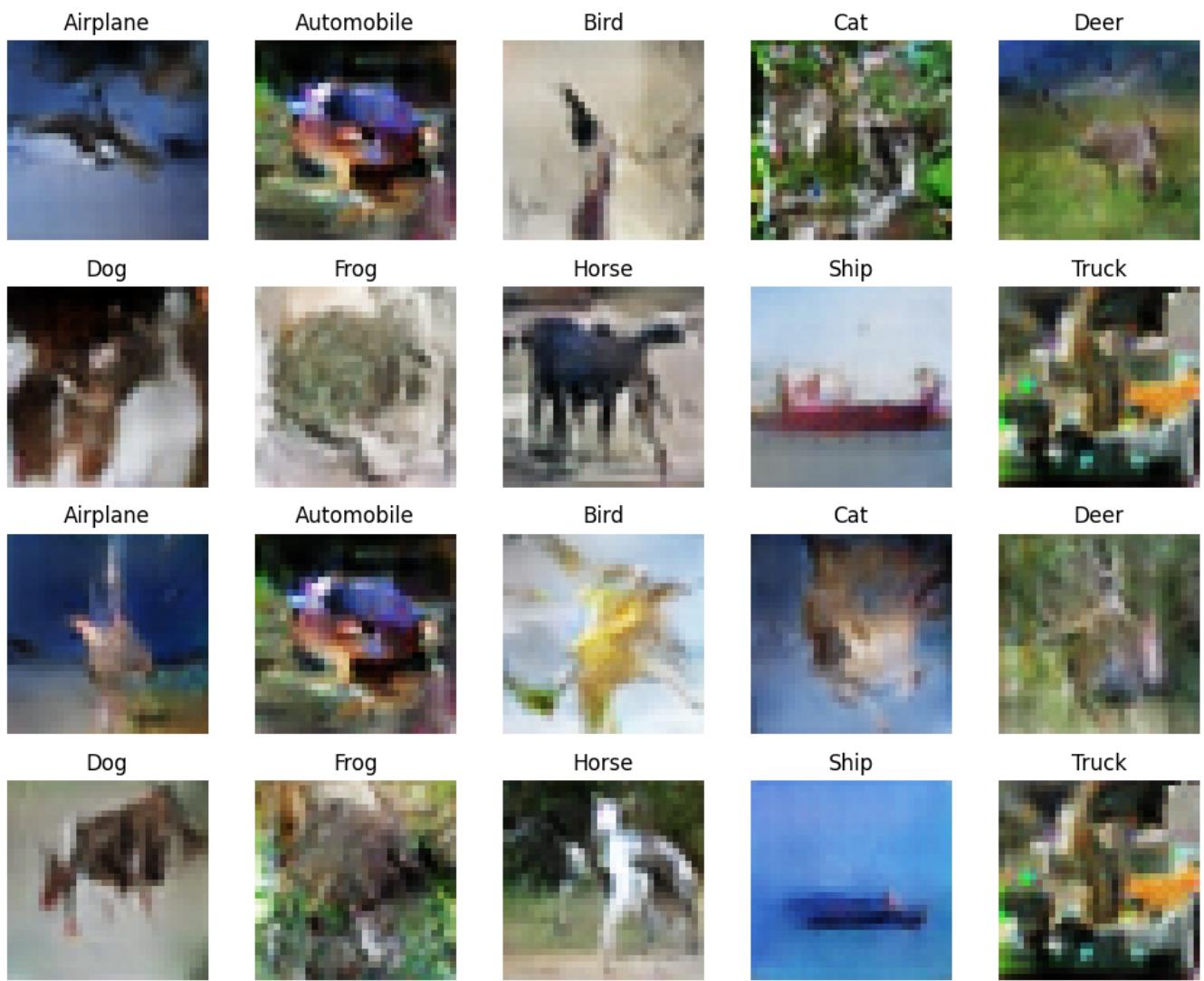
Epoch 169/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0536 - g_loss: 9.3885 - D(x|y): 0.5002 - D(G(z|y)): 0.0240 - KL Divergence: 4.3944

Epoch 170/200

782/782 [=====] - 62s 80ms/step - d_loss: 0.0527 - g_loss: 9.3972 - D(x|y): 0.5002 - D(G(z|y)): 0.0232 - KL Divergence: 4.8576

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - cGAN/generator-epoch-170.h5

Epoch 171/200

782/782 [=====] - 62s 80ms/step - d_loss: 0.0585 - g_loss: 9.3405 - D(x|y): 0.5002 - D(G(z|y)): 0.0260 - KL Divergence: 5.3460

Epoch 172/200

782/782 [=====] - 66s 84ms/step - d_loss: 0.0561 - g_loss: 9.4856 - D(x|y): 0.5004 - D(G(z|y)): 0.0253 - KL Divergence: 4.6888

Epoch 173/200

782/782 [=====] - 62s 79ms/step - d_loss: 0.0562 - g_loss: 9.3107 - D(x|y): 0.5003 - D(G(z|y)): 0.0241 - KL Divergence: 4.5992

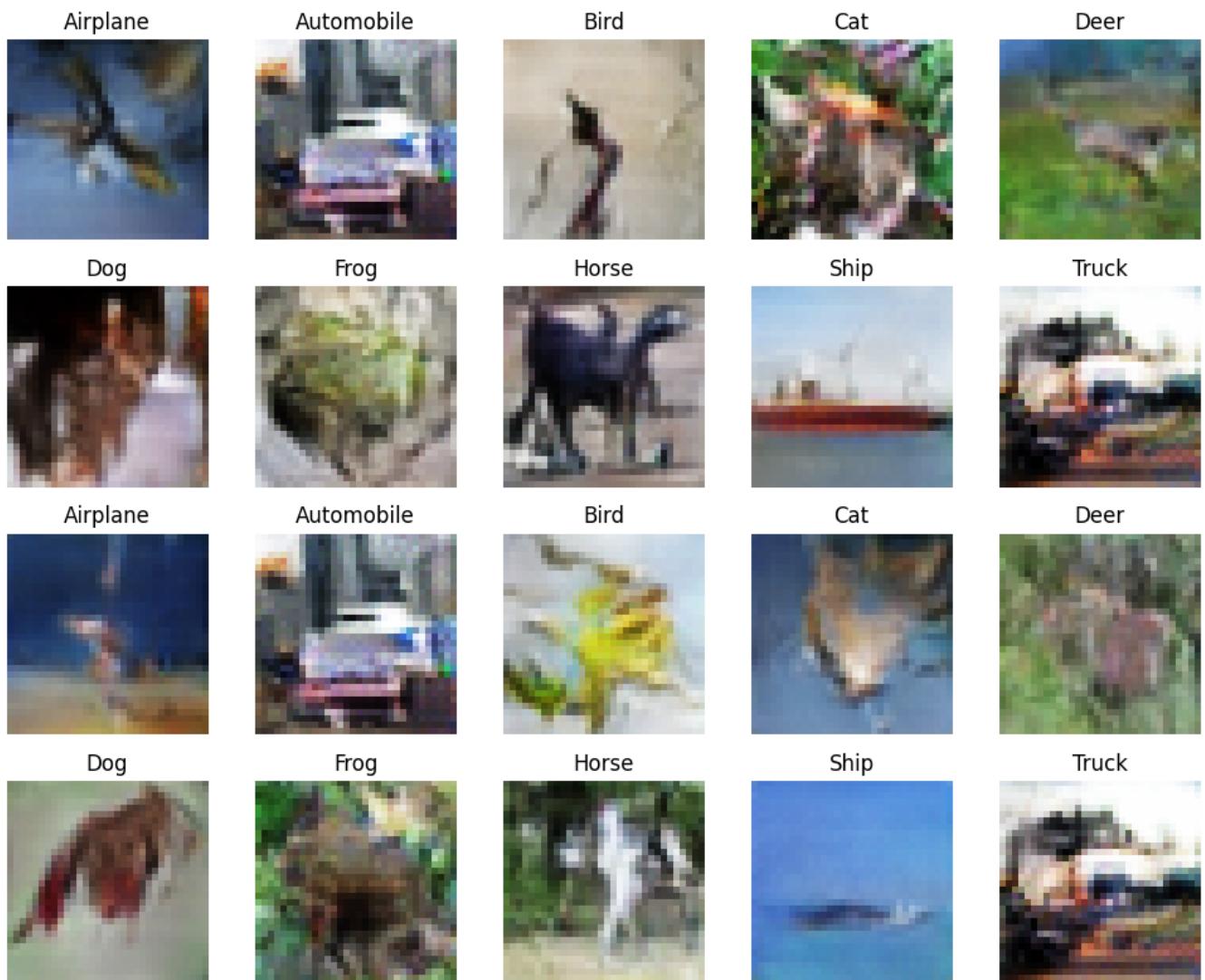
Epoch 174/200

782/782 [=====] - 66s 84ms/step - d_loss: 0.0513 - g_loss: 9.4123 - D(x|y): 0.5000 - D(G(z|y)): 0.0230 - KL Divergence: 4.7673

Epoch 175/200

782/782 [=====] - 63s 80ms/step - d_loss: 0.0531 - g_loss: 9.2189 - D(x|y): 0.5004 - D(G(z|y)): 0.0233 - KL Divergence: 4.8245

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - cGAN/generator-epoch-175.h5

Epoch 176/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0624 - g_loss: 9.4788 - D(x|y): 0.5005 - D(G(z|y)): 0.0268 - KL Divergence: 4.2699

Epoch 177/200

782/782 [=====] - 63s 81ms/step - d_loss: 0.0489 - g_loss: 9.4795 - D(x|y): 0.5000 - D(G(z|y)): 0.0224 - KL Divergence: 4.0231

Epoch 178/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0541 - g_loss: 9.6320 - D(x|y): 0.5002 - D(G(z|y)): 0.0226 - KL Divergence: 4.0148

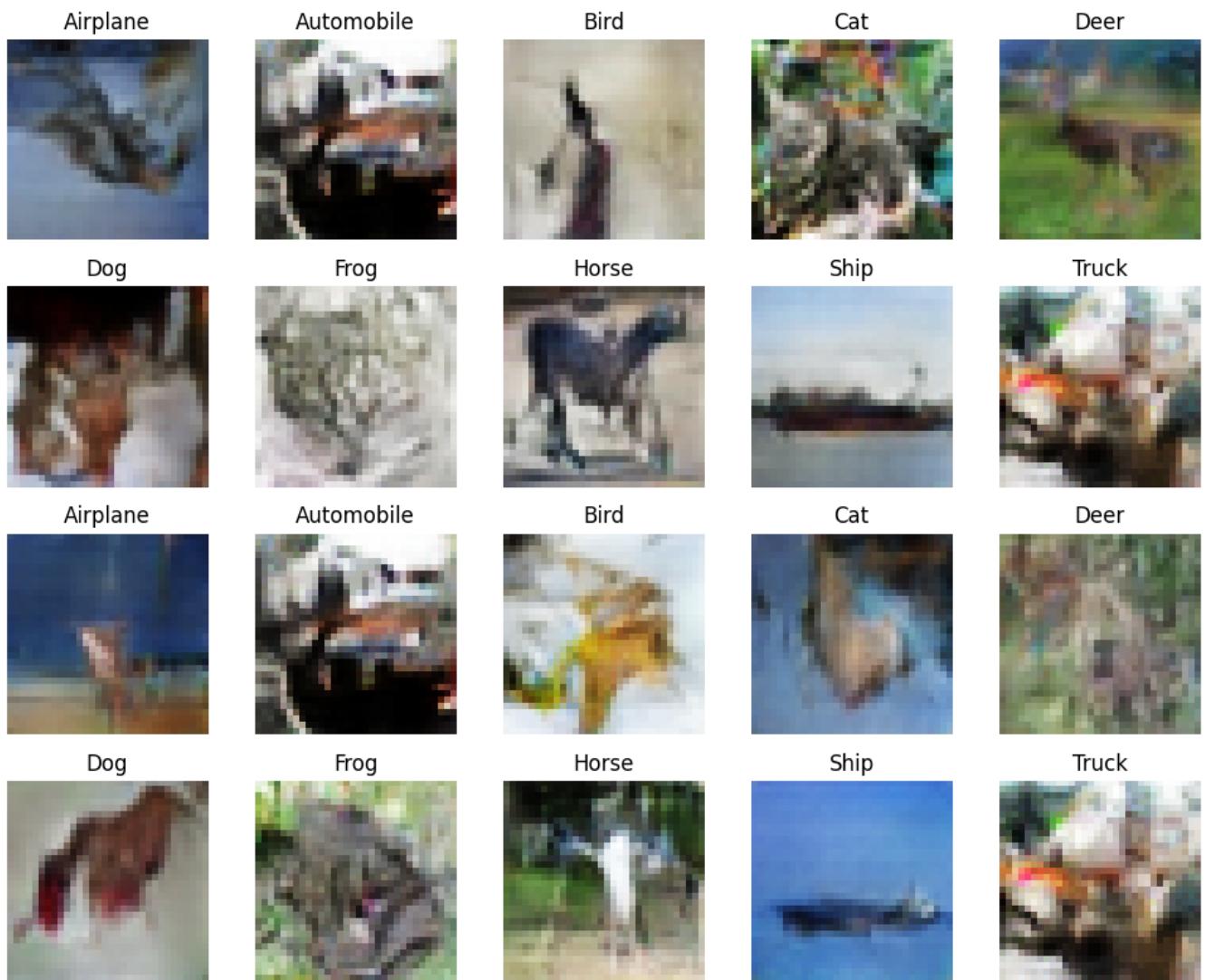
Epoch 179/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.0585 - g_loss: 9.5654 - D(x|y): 0.5005 - D(G(z|y)): 0.0239 - KL Divergence: 4.5631

Epoch 180/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.0527 - g_loss: 9.3711 - D(x|y): 0.4999 - D(G(z|y)): 0.0226 - KL Divergence: 4.5931

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - cGAN/generator-epoch-180.h5

Epoch 181/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.0566 - g_loss: 9.3943 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0237 - KL Divergence: 4.4692

Epoch 182/200

782/782 [=====] - 74s 94ms/step - d_loss: 0.0566 - g_loss: 9.4189 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0227 - KL Divergence: 4.7486

Epoch 183/200

782/782 [=====] - 77s 98ms/step - d_loss: 0.0587 - g_loss: 9.3143 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0255 - KL Divergence: 4.5081

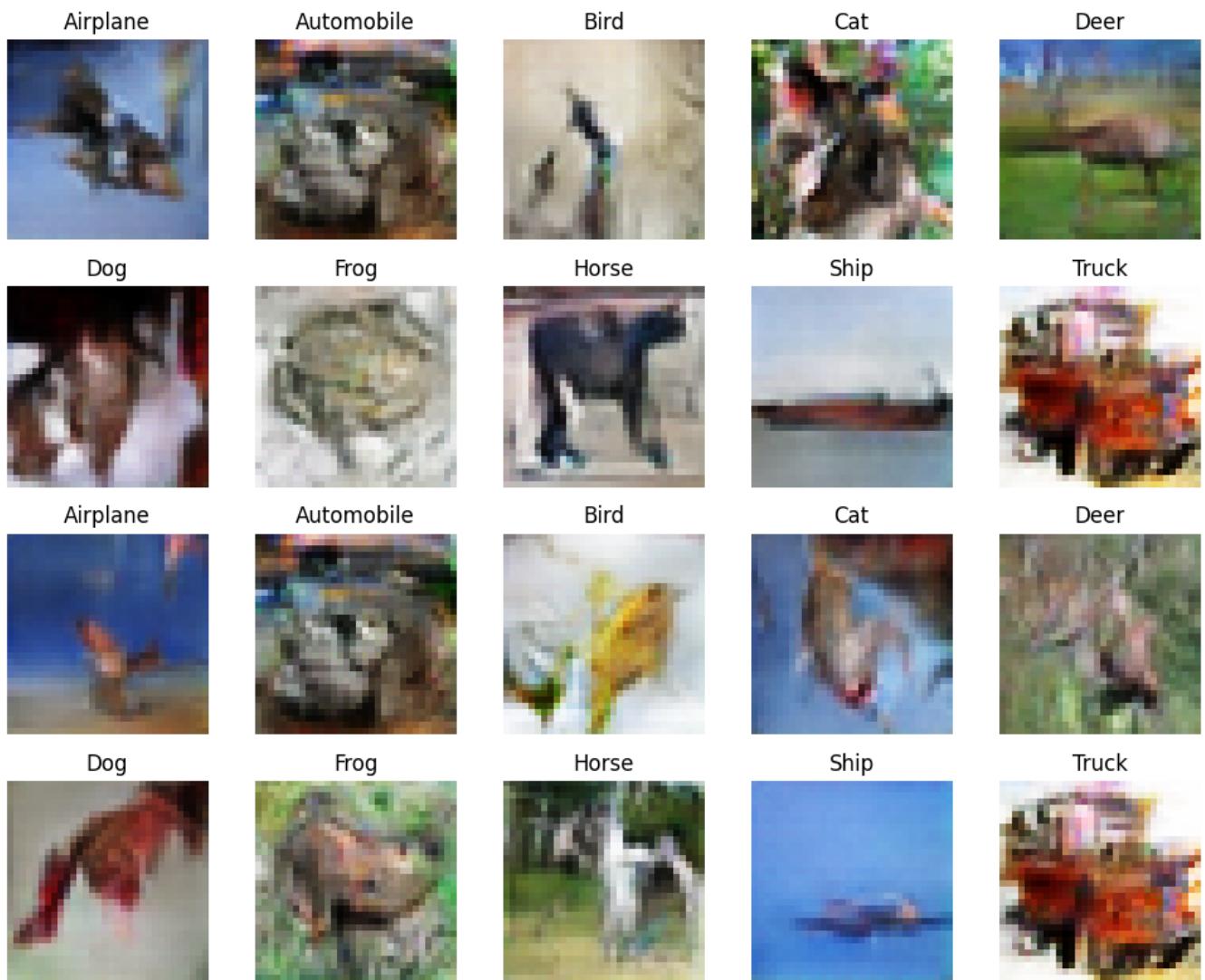
Epoch 184/200

782/782 [=====] - 70s 90ms/step - d_loss: 0.0562 - g_loss: 9.3613 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0254 - KL Divergence: 4.7135

Epoch 185/200

782/782 [=====] - 76s 97ms/step - d_loss: 0.0528 - g_loss: 9.6380 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0232 - KL Divergence: 4.3025

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - cGAN/generator-epoch-185.h5

Epoch 186/200

782/782 [=====] - 79s 102ms/step - d_loss: 0.0624 - g_loss: 8.7457 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0273 - KL Divergence: 4.3775

Epoch 187/200

782/782 [=====] - 88s 113ms/step - d_loss: 0.0487 - g_loss: 9.3505 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0234 - KL Divergence: 4.2476

Epoch 188/200

782/782 [=====] - 93s 119ms/step - d_loss: 0.0526 - g_loss: 9.3762 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0234 - KL Divergence: 4.6776

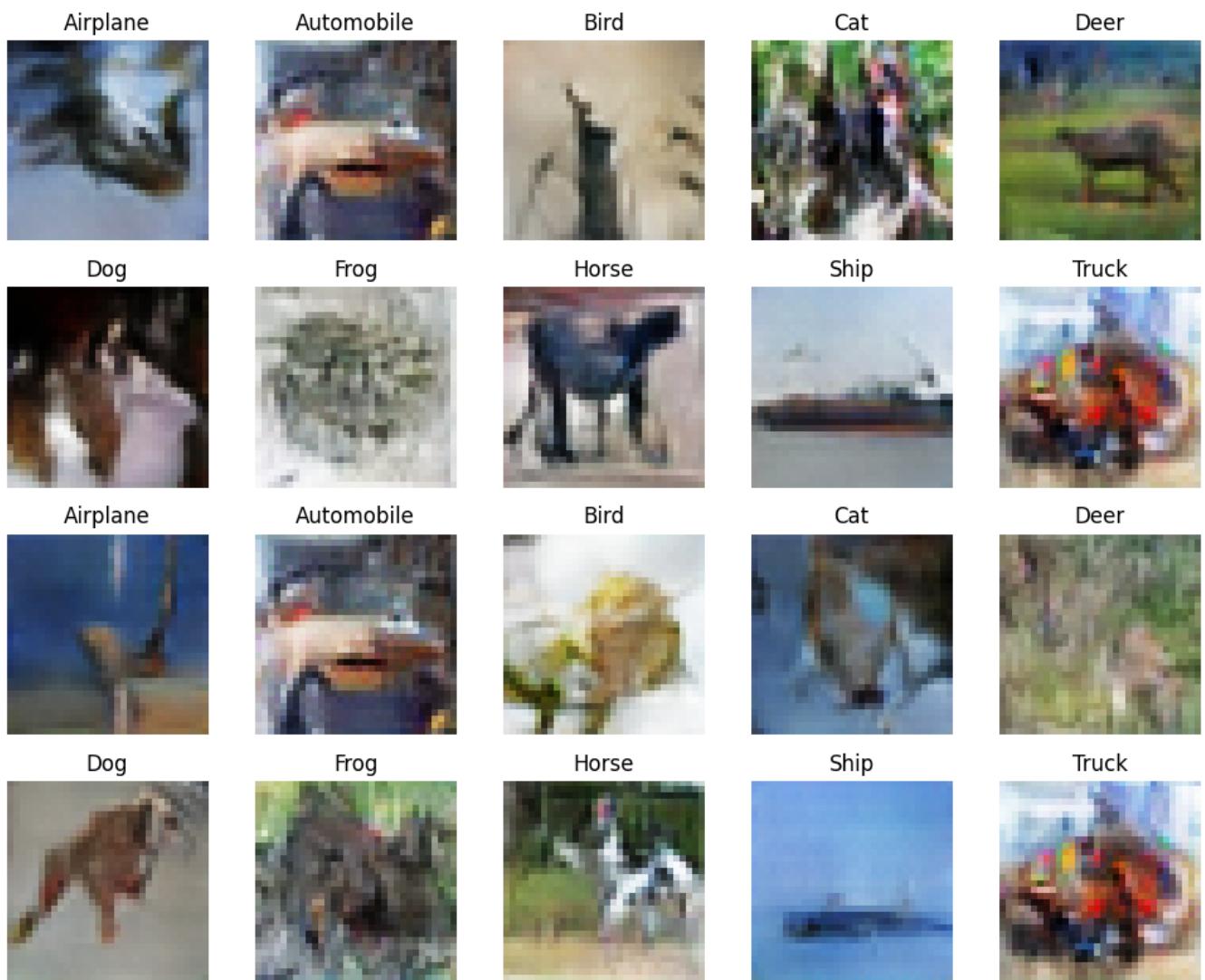
Epoch 189/200

782/782 [=====] - 73s 93ms/step - d_loss: 0.0529 - g_loss: 9.6369 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0231 - KL Divergence: 4.9668

Epoch 190/200

782/782 [=====] - 88s 113ms/step - d_loss: 0.0513 - g_loss: 9.3026 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0224 - KL Divergence: 4.3542

1/1 [=====] - 0s 33ms/step



Generator Checkpoint - cGAN/generator-epoch-190.h5

Epoch 191/200

782/782 [=====] - 86s 109ms/step - d_loss: 0.0515 - g_loss: 9.3558 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0251 - KL Divergence: 4.4981

Epoch 192/200

782/782 [=====] - 71s 91ms/step - d_loss: 0.0554 - g_loss: 9.4918 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0230 - KL Divergence: 4.1773

Epoch 193/200

782/782 [=====] - 84s 108ms/step - d_loss: 0.0611 - g_loss: 9.2813 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0246 - KL Divergence: 4.9039

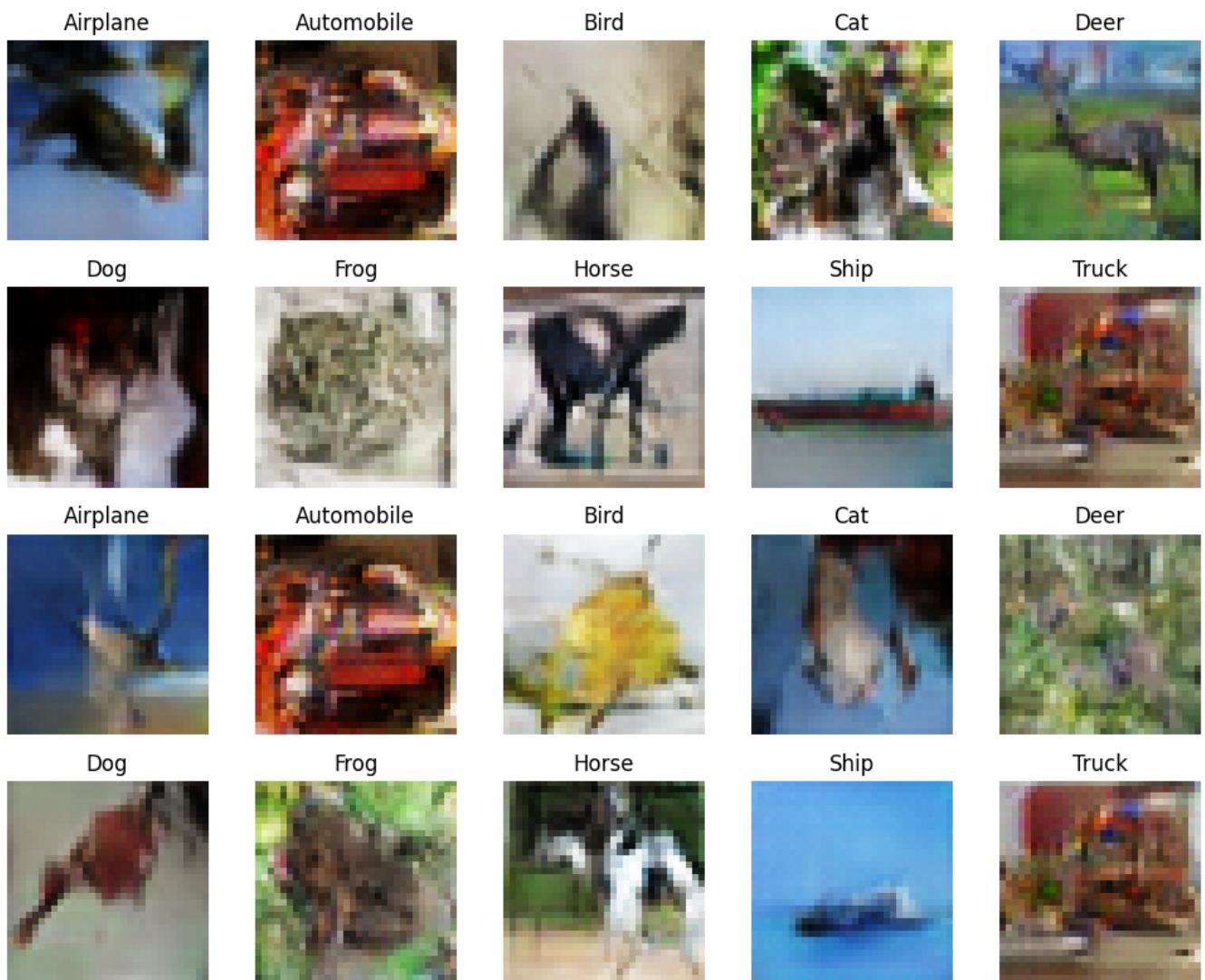
Epoch 194/200

782/782 [=====] - 69s 89ms/step - d_loss: 0.0470 - g_loss: 8.6261 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0234 - KL Divergence: 4.4972

Epoch 195/200

782/782 [=====] - 74s 95ms/step - d_loss: 0.0519 - g_loss: 9.2205 -
 $D(x|y)$: 0.5009 - $D(G(z|y))$: 0.0226 - KL Divergence: 4.6163

1/1 [=====] - 0s 28ms/step



Generator Checkpoint - cGAN/generator-epoch-195.h5

Epoch 196/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.0538 - g_loss: 9.2810 - D(x|y): 0.5002 - D(G(z|y)): 0.0236 - KL Divergence: 4.8282

Epoch 197/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0523 - g_loss: 9.2768 - D(x|y): 0.5004 - D(G(z|y)): 0.0235 - KL Divergence: 4.2958

Epoch 198/200

782/782 [=====] - 76s 97ms/step - d_loss: 0.0522 - g_loss: 9.3172 - D(x|y): 0.5006 - D(G(z|y)): 0.0218 - KL Divergence: 4.7974

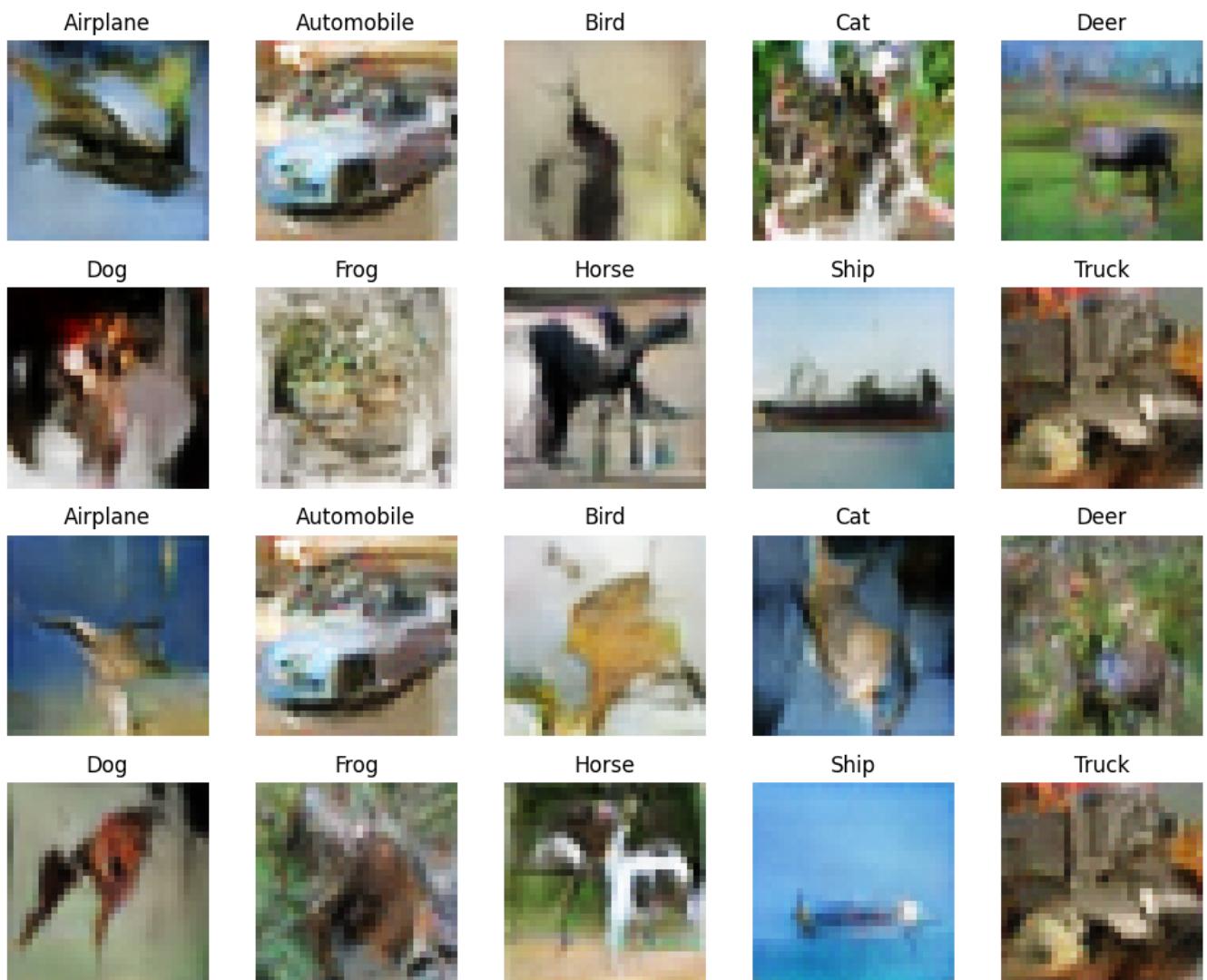
Epoch 199/200

782/782 [=====] - 73s 93ms/step - d_loss: 0.0593 - g_loss: 9.5241 - D(x|y): 0.5004 - D(G(z|y)): 0.0246 - KL Divergence: 4.5065

Epoch 200/200

782/782 [=====] - 73s 93ms/step - d_loss: 0.0547 - g_loss: 9.3898 - D(x|y): 0.5004 - D(G(z|y)): 0.0241 - KL Divergence: 4.4251

1/1 [=====] - 0s 28ms/step



Generator Checkpoint - cGAN/generator-epoch-Full Train.h5

Observations

We notice that around 140 - 145 epochs, the automobile class starts to show the same images despite having the different noises. The model has learned a particular distribution of data which will generate a monotonous output [same images]. After more training certain labels like cats and frog look better and generate better results.

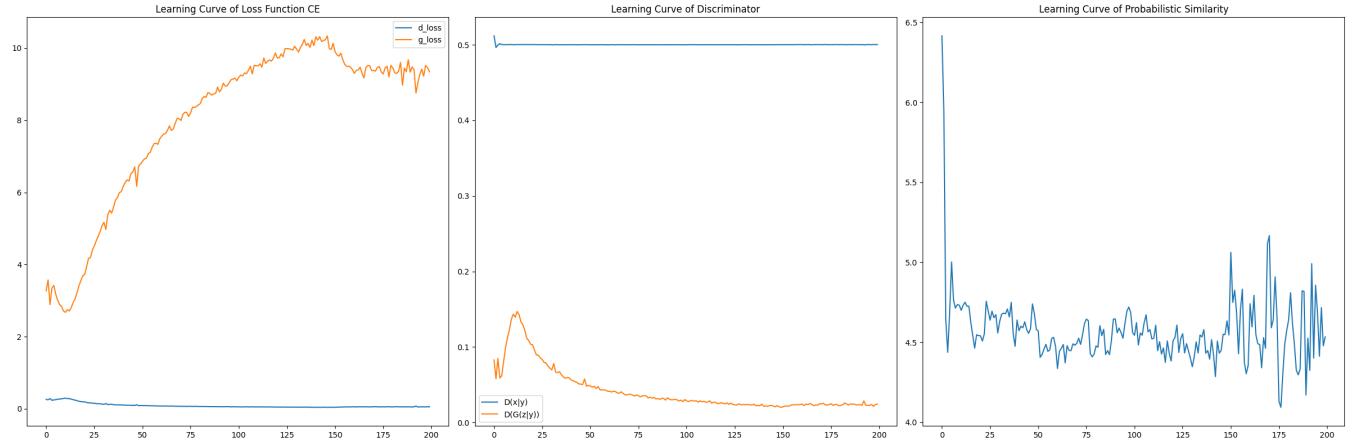
cGAN Evaluation

As mentioned previously we will be using a quantitative and manual evaluation of the cGAN.

We will start off with a quantitative analysis so that we can find the best model to generate the images for manual evaluation.

```
In [ ]: # store history object into dataframe
cond_gan_hist_df = pd.DataFrame(cond_gan_hist.history)

# using pandas dataframe to plot out Learning curve
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(24, 8), tight_layout=True)
cond_gan_hist_df.loc[:, ["d_loss", 'g_loss']].plot(
    ax=ax1, title=r'Learning Curve of Loss Function CE')
cond_gan_hist_df.loc[:, ['D(x|y)', 'D(G(z|y))']].plot(
    ax=ax2, title=r'Learning Curve of Discriminator')
cond_gan_hist_df.loc[:, 'KL Divergence'].plot(
    ax=ax3, title=r'Learning Curve of Probabilistic Similarity')
plt.show()
```



Observations

We notice that around 140 - 145 epochs, the model starts collapsing where the images start repeating. This is seen from the learning curve where the `g_loss` started dropping. We also note that after 145 epochs the KL divergences starts fluctuating rapidly. To choose best model, we will choose the lowest KL divergence before 140 epochs which is 130 epochs.

```
In [ ]: # Loading Weights for best Generator
saved_weights = 'cGAN/generator-epoch-130.h5'
cond_gan.generator.load_weights(saved_weights)
cond_gan.generator.summary()
```

Model: "generator_cGAN"

Layer (type)	Output Shape	Param #	Connected to
=====			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
=====			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_5 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_7 (Dense)	(None, 2048)	284672	['concatenate_5[0][0]']
batch_normalization_120 (Batch Normalization)	(None, 2048)	8192	['dense_7[0][0]']
leaky_re_lu_20 (LeakyReLU)	(None, 2048)	0	['batch_normalization_120[0][0]']
reshape_3 (Reshape)	(None, 2, 2, 512)	0	['leaky_re_lu_20[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2754752	['reshape_3[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
=====			
=====			
Total params: 3,050,691			
Trainable params: 3,045,699			
Non-trainable params: 4,992			

In []:

```
n = 10000

# generating labels
labels = np.random.randint(low=0, high=10, size=n)
one_hot_labels = to_categorical(labels)

# Generating 1000 Synthetic Images
random_noise = tf.random.normal(shape=(n, NOISE))

synthetic_images = cond_gan.generator.predict(
    [random_noise, one_hot_labels])
print("Latent Vector Dim: {} \t Generated Images Dim: {}".format(
    random_noise.shape, synthetic_images.shape))

# Scaling back to [0, 1]
synthetic_images -= -1
synthetic_images /= (1 - (-1))

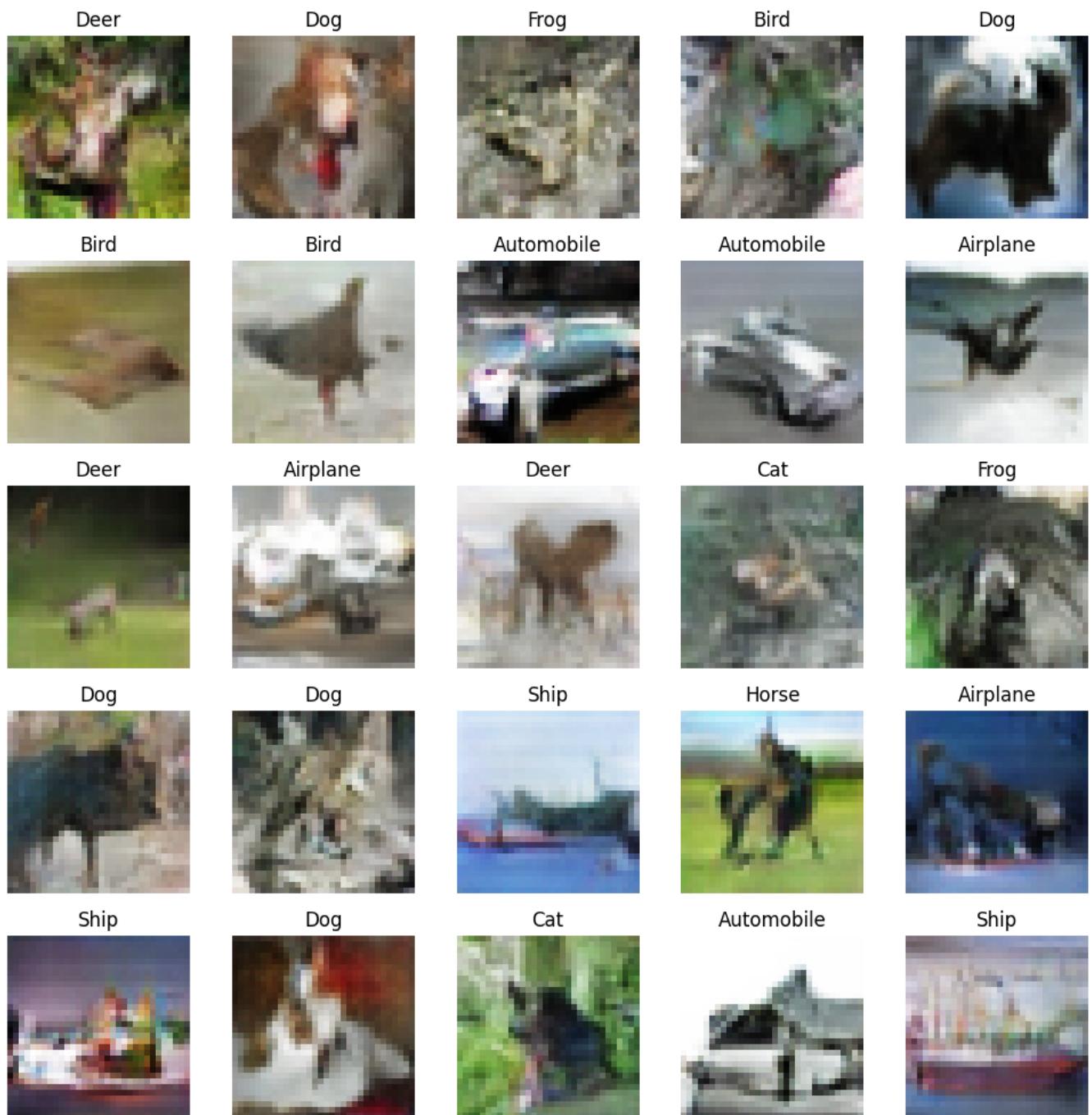
# Display 25 randomly sampled images
```

```

fig = plt.figure(figsize=(10, 10), tight_layout=True)
for i in range(25):
    rand_idx = np.random.randint(0, len(synthetic_images))
    ax = fig.add_subplot(5, 5, i+1)
    ax.imshow(synthetic_images[rand_idx])
    ax.set_title(class_labels[labels[rand_idx]])
    ax.axis('off')
plt.show()

```

313/313 [=====] - 3s 7ms/step
Latent Vector Dim: (10000, 128) Generated Images Dim: (10000, 32, 32, 3)



Observations

We can see that the model is able to clearly generate vehicles with good precision but the animals are more jumbled up. Features like the texture can be seen in some of the images which suggest the model should want to have more training to help make the model better. But due to model collapsing, changes needed to be made to the architecture and using dropout and regularisation methods can make the model better.

In []: cond_gan_fid_class = GAN_FID(batch_size=512, noise=128,
sample_size=10000, buffer_size=1024, labels=True)

```

cond_gan_fid_class.fit(generator=cond_gan.generator, train_data=x_train)
fid_score = cond_gan_fid_class.evaluate()

Computing Real Image Embeddings
 0%|          | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 4s 133ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 134ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 134ms/step
16/16 [=====] - 2s 133ms/step

Computing Generated Image Embeddings
 0%|          | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 153ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 144ms/step
16/16 [=====] - 2s 155ms/step
16/16 [=====] - 3s 161ms/step
16/16 [=====] - 3s 159ms/step
16/16 [=====] - 3s 159ms/step
16/16 [=====] - 3s 159ms/step
16/16 [=====] - 3s 162ms/step
16/16 [=====] - 3s 168ms/step
16/16 [=====] - 3s 160ms/step
16/16 [=====] - 3s 158ms/step
16/16 [=====] - 3s 159ms/step
16/16 [=====] - 3s 164ms/step

Computed Embeddings      Real Images Embedding Shape: (10240, 2048)      Generated Images Embe
dding Shape: (10240, 2048)
The computed FID score is: 94.536161025068

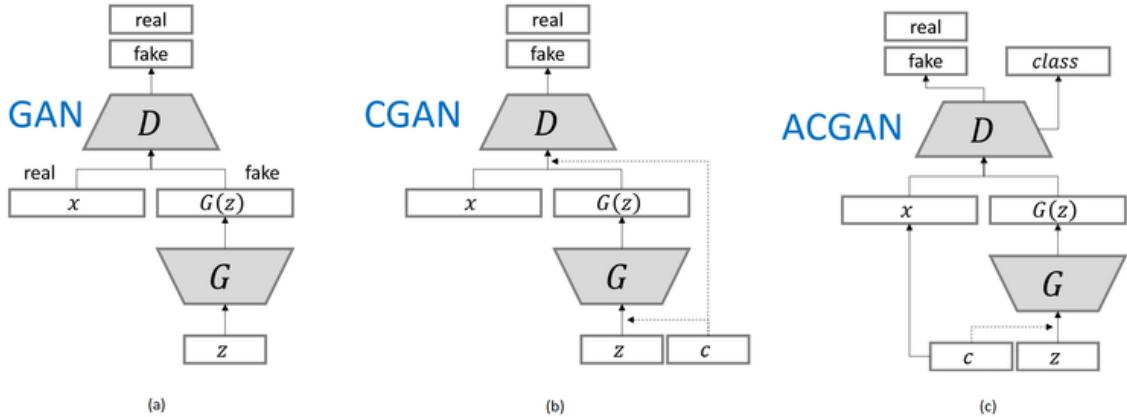
```

Observations

We see that the FID score has significantly reduced from 153.4580126583096 to 94.536161025068 [DCGAN to cGAN]. Although the FID score is still high but the score is below 100, we can see that with the addition of labels the model is more closely resembling the training data but there are still some noticeable differences.

ACGAN

Auxiliary Classifier Generative Adversarial Network, or ACGAN in short, follows a similar architecture to a Conditional GAN. There are two main difference between the two architectures - the discriminator produces two outputs and an auxiliary loss function is included in the training process.



Therefore, we will be only modifying the discriminator.

ACGAN Discriminator

As the ACGAN is very similar to the cGAN, we will be only modifying the last output layer. The ACGAN wants to not only lower the loss between real and fake images, but also reduce the loss between the conditions and the predicted conditions.

```
In [ ]: # function to create discriminator model
def create_ACGAN_discriminator(image_size):
    # input image
    img_input = Input(shape=(image_size), name='Image_Input')

    # conditions y
    conditions = Input(shape=(10,), name='Conditions_y')

    base_discriminator = Sequential([
        Conv2D(64, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
        Conv2D(128, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
        Conv2D(256, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
        Conv2D(512, kernel_size=4, strides=2, padding='same'),
        BatchNormalization(momentum=0.9),
        LeakyReLU(alpha=0.1),
    ], name='Base_Discriminator')
    discriminator = base_discriminator(img_input)

    discriminator = GlobalAveragePooling2D()(discriminator)

    # Concatenate - combine with conditions y
    merged_layer = Concatenate()([discriminator, conditions])
    discriminator = Dense(512, activation='relu')(merged_layer)

    # Output
    Disc_Output_Layer = Dense(1, activation='sigmoid',
                               name='Disc_Output_Layer')(discriminator)
    Aux_Output_Layer = Dense(10, activation='softmax',
                             name='Aux_Output_Layer')(discriminator)

    discriminator = Model(inputs=[img_input, conditions],
                          outputs=[Disc_Output_Layer, Aux_Output_Layer], name='discriminator')

    return discriminator

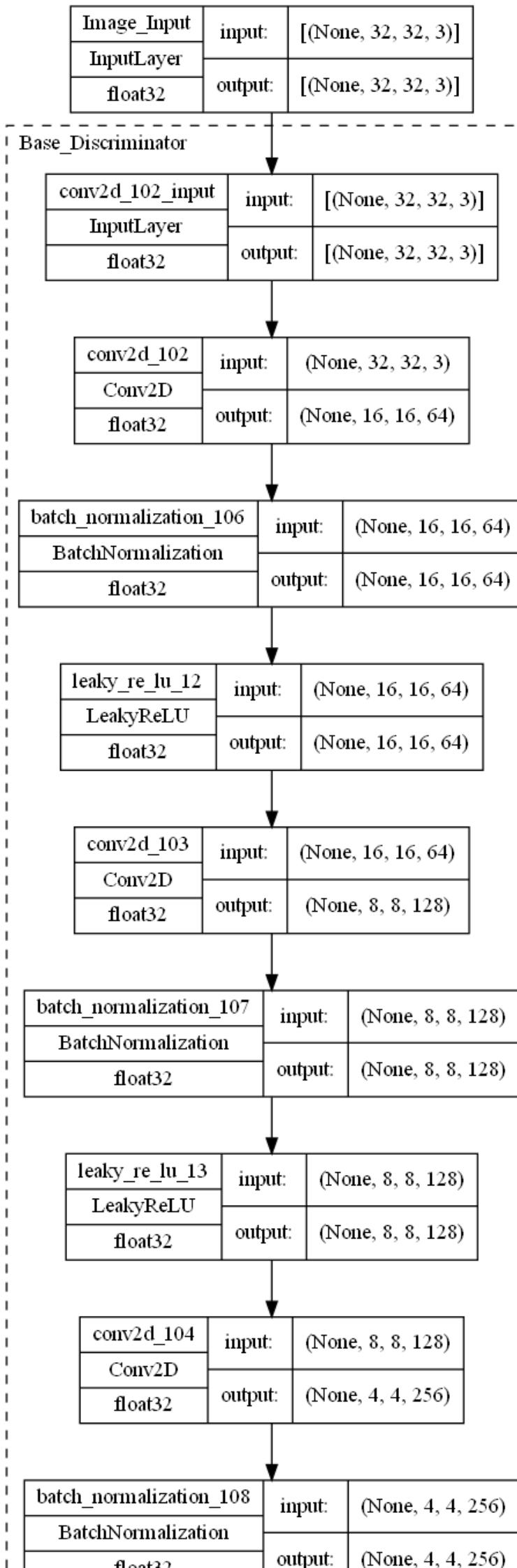
create_ACGAN_discriminator(image_size=IMG_SIZE).summary()
```

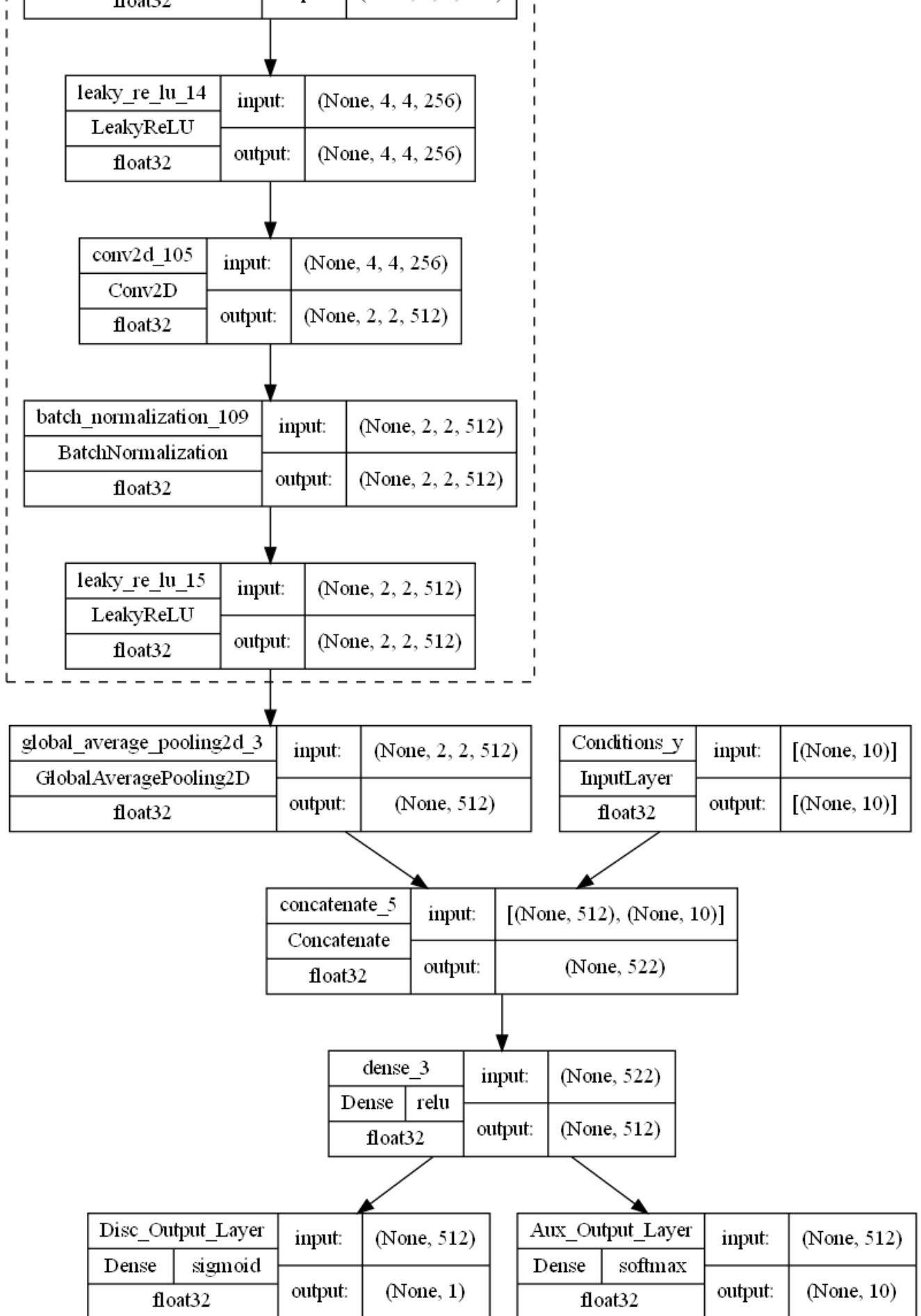
Model: "discriminator_ACGAN"

Layer (type)	Output Shape	Param #	Connected to
=====			
Image_Input (InputLayer)	[(None, 32, 32, 3)]	0	[]
=====			
Base_Discriminator (Sequential (None, 2, 2, 512))	(None, 2, 2, 512)	2760384	['Image_Input[0][0]']
=====			
Image_Input (InputLayer)	[(None, 32, 32, 3)]	0	[]
Base_Discriminator (Sequential (None, 2, 2, 512))	(None, 2, 2, 512)	2760384	['Image_Input[0][0]']
global_average_pooling2d_8 (G1 (None, 512) obalAveragePooling2D)	(None, 512)	0	['Base_Discriminator[0][0]']
Conditions_y (InputLayer)	[(None, 10)]	0	[]
concatenate_14 (Concatenate [0][0]	(None, 522)	0	['global_average_pooling2d_8 [0][0]', 'Conditions_y[0][0]']
dense_12 (Dense)	(None, 512)	267776	['concatenate_14[0][0]']
Disc_Output_Layer (Dense)	(None, 1)	513	['dense_12[0][0]']
Aux_Output_Layer (Dense)	(None, 10)	5130	['dense_12[0][0]']
=====			
Total params: 3,033,803			
Trainable params: 3,031,883			
Non-trainable params: 1,920			

In []: plot_model(create_ACGAN_discriminator(image_size=IMG_SIZE), to_file="images/models/ACGAN_disc", show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True)

Out[]:





ACGAN Training Process

It is similar to the cGAN Training process but with a new loss function. To help us with the code, we will be referring to Keras example on a conditional GAN.

https://github.com/keras-team/keras-io/blob/master/examples/generative/conditional_gan.py

```
In [ ]: class AConditionalGAN(tf.keras.Model):
    def __init__(self, discriminator, generator, noise):
        super(AConditionalGAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.noise = noise
        self.gen_loss_tracker = tf.keras.metrics.Mean(name="generator_loss")
        self.disc_loss_tracker = tf.keras.metrics.Mean(
            name="discriminator_loss")
        self.d_xy_tracker = tf.keras.metrics.Mean(name='Mean D(x|y)')
        self.d_g_zy_tracker = tf.keras.metrics.Mean(name='Mean D(G(z|y))')
        self.kl = tf.keras.metrics.KLDivergence()

    def compile(self, d_optimizer, g_optimizer, disc_loss_fn, aux_loss_fn):
        super(AConditionalGAN, self).compile()
        self.d_optimizer = d_optimizer
        self.g_optimizer = g_optimizer
        self.disc_loss_fn = disc_loss_fn
        self.aux_loss_fn = aux_loss_fn

    def train_step(self, data):
        ### TRAINING DISCRIMINATOR ###
        # Unpack the data.
        real_images, condition = data

        # Sample for Latent noise vector z
        batch_size = tf.shape(real_images)[0]
        latent_noise_vector = tf.random.normal(
            shape=(batch_size, self.noise))

        # Maps the noise Latent vector and Labels to generate fake images.
        generated_images = self.generator([latent_noise_vector, condition])

        # Combine with real images
        combined_images = tf.concat([generated_images, real_images], axis=0)
        combined_condition = tf.concat([condition, condition], axis=0)

        # Discrimination
        labels = tf.concat(
            [tf.zeros((batch_size, 1)), tf.ones((batch_size, 1))], axis=0
        )

        # Train the discriminator.
        with tf.GradientTape() as tape:
            disc_output, aux_output = self.discriminator(
                [combined_images, combined_condition])
            disc_d_loss = self.disc_loss_fn(labels, disc_output)
            aux_d_loss = self.aux_loss_fn(combined_condition, aux_output)
            d_loss = disc_d_loss + aux_d_loss
            grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
            self.d_optimizer.apply_gradients(
                zip(grads, self.discriminator.trainable_weights)
            )

        # Computing D(x/y)
        d_xy = tf.math.reduce_mean(disc_output)

        ### TRAINING GENERATOR ###
        latent_noise_vector = tf.random.normal(
            shape=(batch_size, self.noise))

        # Assemble Labels that say "all real images".
        misleading_labels = tf.ones((batch_size, 1))

        # Train the Generator
```

```

    with tf.GradientTape() as tape:
        fake_images = self.generator([latent_noise_vector, condition])
        disc_output, aux_output = self.discriminator(
            [fake_images, condition])
        disc_g_loss = self.disc_loss_fn(misleading_labels, disc_output)
        aux_g_loss = self.aux_loss_fn(condition, aux_output)
        g_loss = disc_g_loss + aux_g_loss
        grads = tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(
            zip(grads, self.generator.trainable_weights))

    # Computing D(G(z|y))
    d_g_zy = tf.math.reduce_mean(disc_output)

    # Monitor loss and metrics.
    self.gen_loss_tracker.update_state(g_loss)
    self.disc_loss_tracker.update_state(d_loss)
    self.d_xy_tracker.update_state(d_xy)
    self.d_g_zy_tracker.update_state(d_g_zy)
    self.kl.update_state(real_images, fake_images)

    return {
        "d_loss": self.disc_loss_tracker.result(),
        "g_loss": self.gen_loss_tracker.result(),
        "D(x|y)": self.d_xy_tracker.result(),
        "D(G(z|y))": self.d_g_zy_tracker.result(),
        "KL Divergence": self.kl.result(),
    }
}

```

ACGAN Callback

To help monitor the ACGAN's loss and progress and visualise the images after every patience. We will need to make a custom tensorflow callback that will help monitor the ACGAN.

```

In [ ]: class ACGANMonitor(keras.callbacks.Callback):
    def __init__(self, num_img=20, noise=128, patience=10, vmin=0, vmax=1):
        self.num_img = num_img
        self.noise = noise
        self.patience = patience
        self.vmin = vmin
        self.vmax = vmax
        self.latent_noise_vector = tf.random.normal(
            shape=(self.num_img, self.noise))
        self.conditions = to_categorical([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                                         0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

    def generate_plot(self):
        # Generate Images
        generated_images = self.model.generator.predict(
            [self.latent_noise_vector, self.conditions])
        # Normalise Image from [vmin, vmax] to [0, 1]
        generated_images -= self.vmin
        generated_images /= (self.vmax - self.vmin)
        row_size = int(np.ceil(self.num_img/5))
        fig = plt.figure(figsize=(10, 2*row_size), tight_layout=True)
        for i in range(self.num_img):
            ax = fig.add_subplot(row_size, 5, i+1)
            ax.imshow(generated_images[i])
            ax.set_title(class_labels[i % 10])
            ax.axis('off')
        plt.show()

    def save_weights(self, epoch=None):
        try:
            if epoch != None:

```

```

        name = 'ACGAN/generator-epoch-{}.h5'.format(epoch)
        print('Generator Checkpoint - {}'.format(name))
        self.model.generator.save_weights(
            filepath=name,
            save_format='h5'
        )
    except Exception as e:
        print(e)

    def on_epoch_begin(self, epoch, logs=None):
        if epoch % self.patience == 0:
            self.generate_plot()
            self.save_weights(epoch)

    def on_train_end(self, epoch, logs=None):
        self.generate_plot()
        self.save_weights('Full Train')

```

```
In [ ]: callbacks = [
    ACGANMonitor(num_img=20, noise=128, patience=5, vmin=-1, vmax=1),
]
```

ACGAN Training

After setting the ACGAN architecture, we can now run the ACGAN models to generate images. First, we need to convert the data into a tensorflow dataset and training using multiprocessing to speed up process.

```

In [ ]: tf.keras.backend.clear_session()
K.clear_session()

aux_cond_gan = AConditionalGAN(
    discriminator=create_ACGAN_discriminator(image_size=IMG_SIZE),
    generator=create_cGAN_generator(noise=NOISE),
    noise=NOISE
)

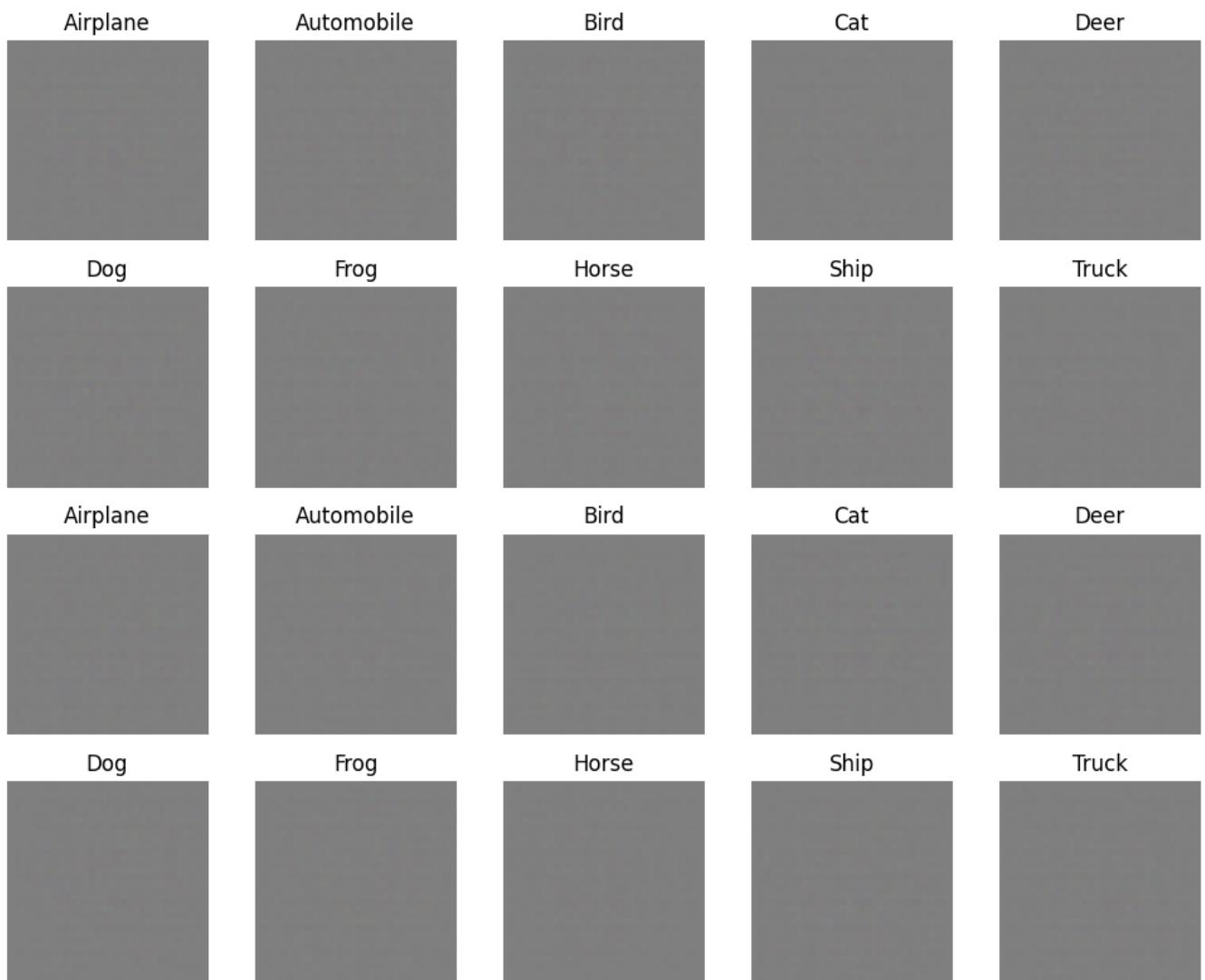
aux_cond_gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    disc_loss_fn=keras.losses.BinaryCrossentropy(),
    aux_loss_fn=keras.losses.CategoricalCrossentropy()
)

dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
dataset = dataset.shuffle(buffer_size=1024).batch(
    BATCH_SIZE, num_parallel_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)

aux_cond_hist = aux_cond_gan.fit(
    dataset, epochs=200, use_multiprocessing=True, workers=16, callbacks=callbacks)

```

1/1 [=====] - 0s 371ms/step



Generator Checkpoint - ACGAN/generator-epoch-0.h5

Epoch 1/200

782/782 [=====] - 75s 90ms/step - d_loss: 1.6720 - g_loss: 4.5785 -
 $D(x|y)$: 0.5349 - $D(G(z|y))$: 0.1203 - KL Divergence: 5.3958

Epoch 2/200

782/782 [=====] - 66s 85ms/step - d_loss: 0.3243 - g_loss: 3.9786 -
 $D(x|y)$: 0.4951 - $D(G(z|y))$: 0.0453 - KL Divergence: 4.2473

Epoch 3/200

782/782 [=====] - 66s 85ms/step - d_loss: 0.2607 - g_loss: 3.6651 -
 $D(x|y)$: 0.5007 - $D(G(z|y))$: 0.0535 - KL Divergence: 5.9017

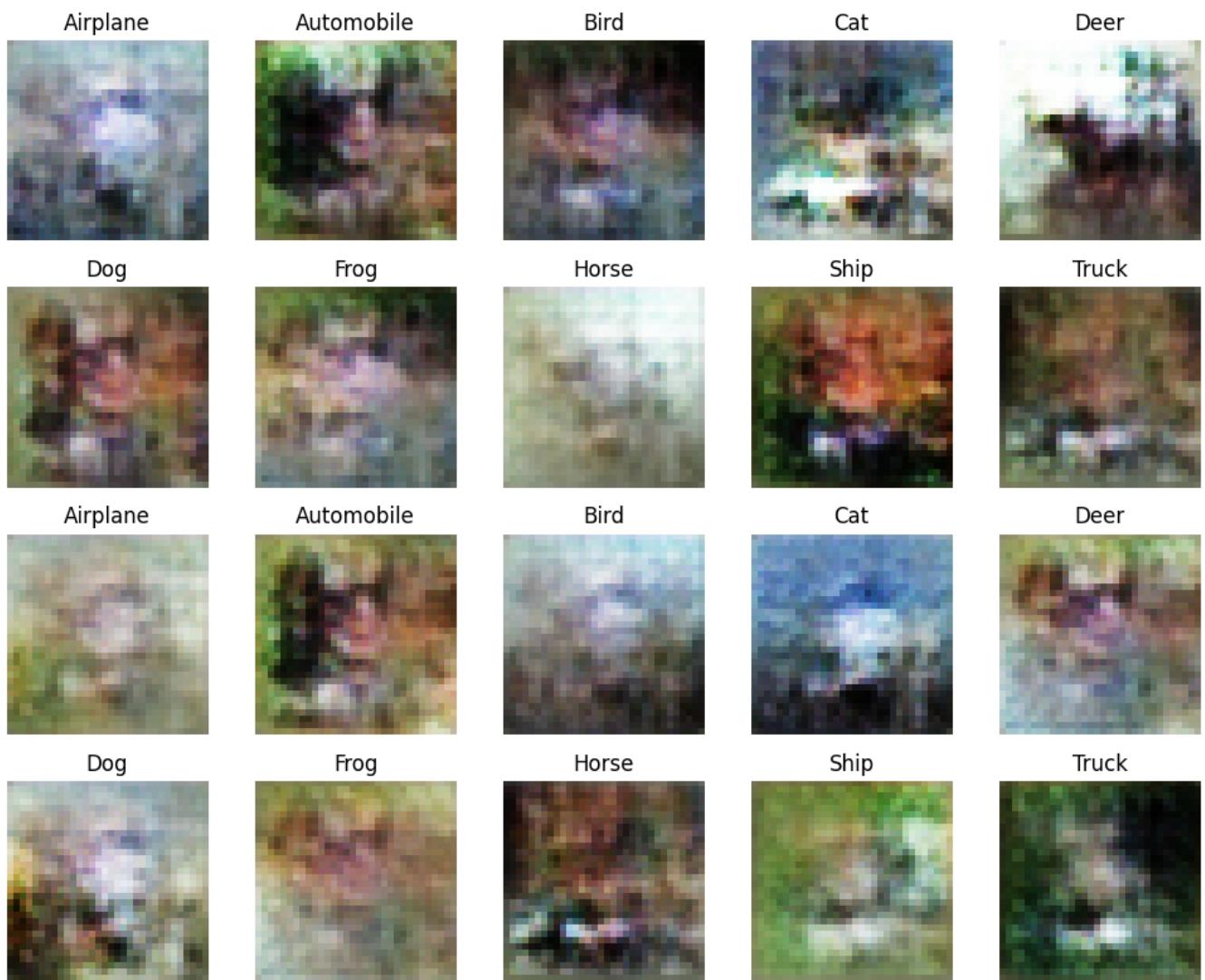
Epoch 4/200

782/782 [=====] - 66s 85ms/step - d_loss: 0.2814 - g_loss: 3.2607 -
 $D(x|y)$: 0.5010 - $D(G(z|y))$: 0.0685 - KL Divergence: 5.0985

Epoch 5/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2491 - g_loss: 3.5809 -
 $D(x|y)$: 0.4996 - $D(G(z|y))$: 0.0565 - KL Divergence: 5.2626

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-5.h5

Epoch 6/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.2577 - g_loss: 3.2986 - D(x|y): 0.4998 - D(G(z|y)): 0.0740 - KL Divergence: 5.2817

Epoch 7/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.2724 - g_loss: 3.1666 - D(x|y): 0.5004 - D(G(z|y)): 0.0924 - KL Divergence: 5.5191

Epoch 8/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.2407 - g_loss: 3.2719 - D(x|y): 0.5004 - D(G(z|y)): 0.0859 - KL Divergence: 5.2422

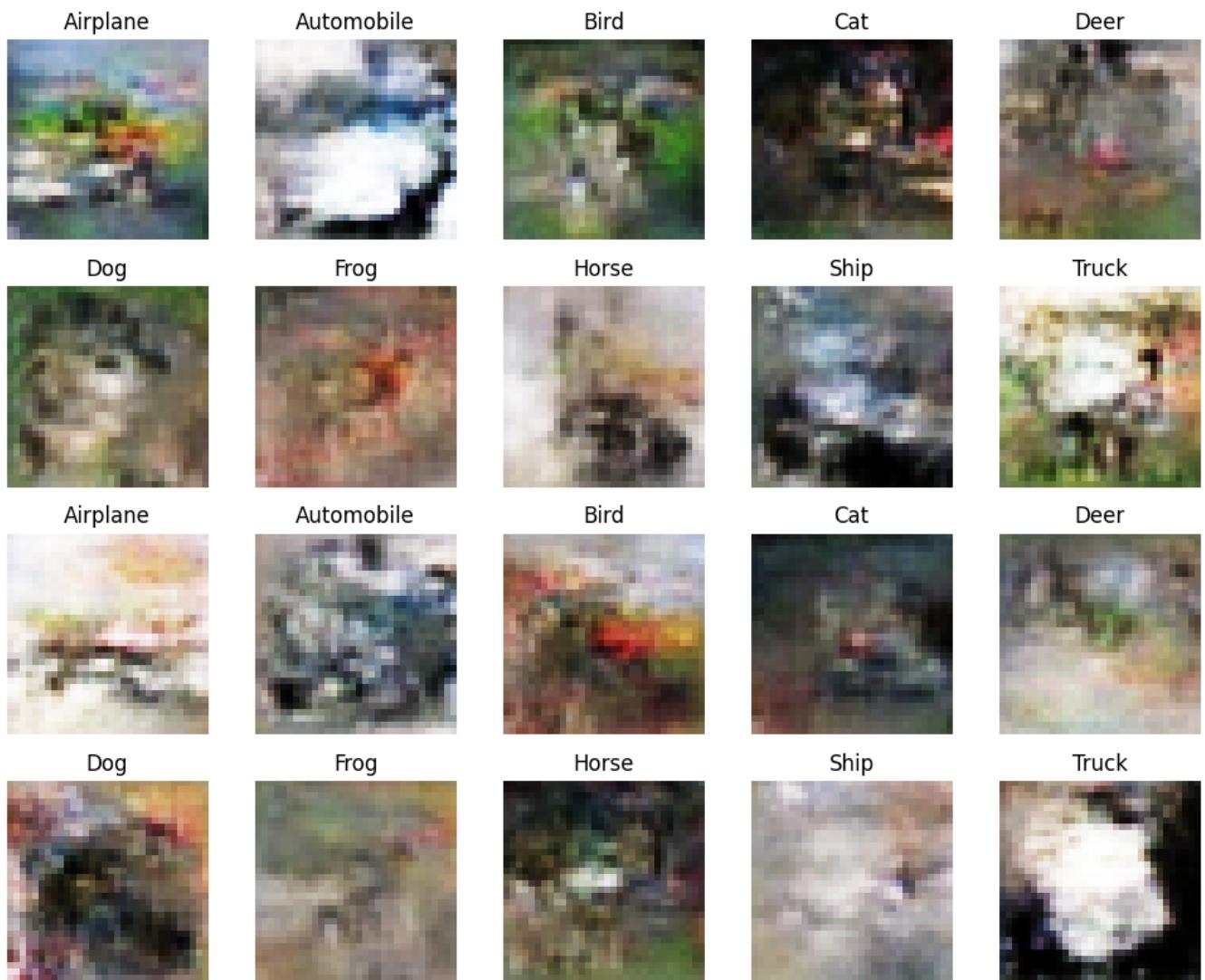
Epoch 9/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2315 - g_loss: 3.2485 - D(x|y): 0.5009 - D(G(z|y)): 0.0922 - KL Divergence: 5.2537

Epoch 10/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2320 - g_loss: 3.3117 - D(x|y): 0.4987 - D(G(z|y)): 0.0934 - KL Divergence: 5.1520

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - ACGAN/generator-epoch-10.h5

Epoch 11/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2672 - g_loss: 3.0352 - D(x|y): 0.4992 - D(G(z|y)): 0.1167 - KL Divergence: 5.0254

Epoch 12/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.2477 - g_loss: 3.0386 - D(x|y): 0.5004 - D(G(z|y)): 0.1165 - KL Divergence: 4.9043

Epoch 13/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2460 - g_loss: 3.0714 - D(x|y): 0.5000 - D(G(z|y)): 0.1165 - KL Divergence: 4.7482

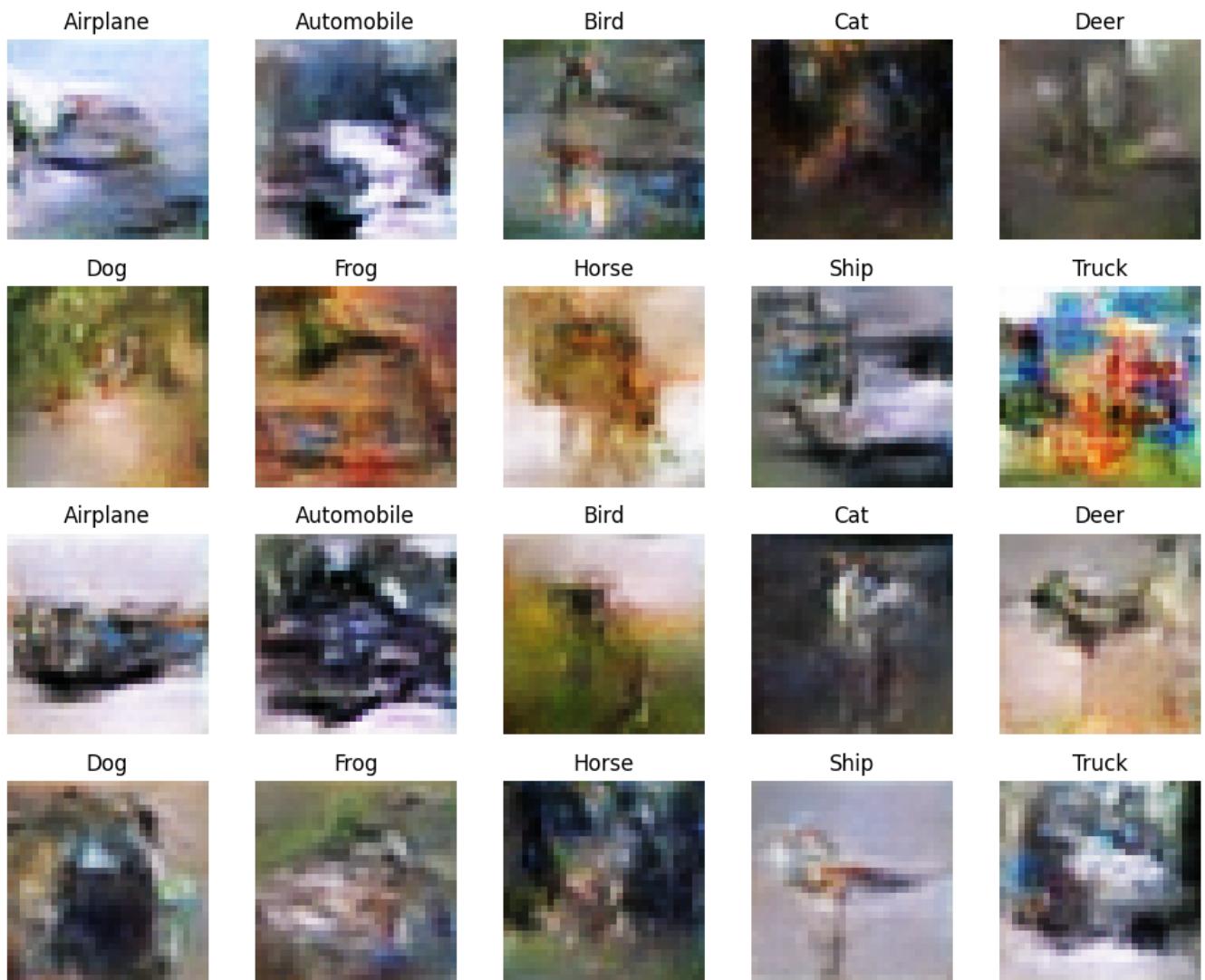
Epoch 14/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.2442 - g_loss: 3.0982 - D(x|y): 0.5004 - D(G(z|y)): 0.1158 - KL Divergence: 4.7992

Epoch 15/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.2402 - g_loss: 3.0818 - D(x|y): 0.5001 - D(G(z|y)): 0.1187 - KL Divergence: 4.7124

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - ACGAN/generator-epoch-15.h5

Epoch 16/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.2383 - g_loss: 3.1354 - D(x|y): 0.4996 - D(G(z|y)): 0.1175 - KL Divergence: 4.6767

Epoch 17/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.2391 - g_loss: 3.0784 - D(x|y): 0.5007 - D(G(z|y)): 0.1222 - KL Divergence: 4.6464

Epoch 18/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2368 - g_loss: 3.1370 - D(x|y): 0.5005 - D(G(z|y)): 0.1213 - KL Divergence: 4.5927

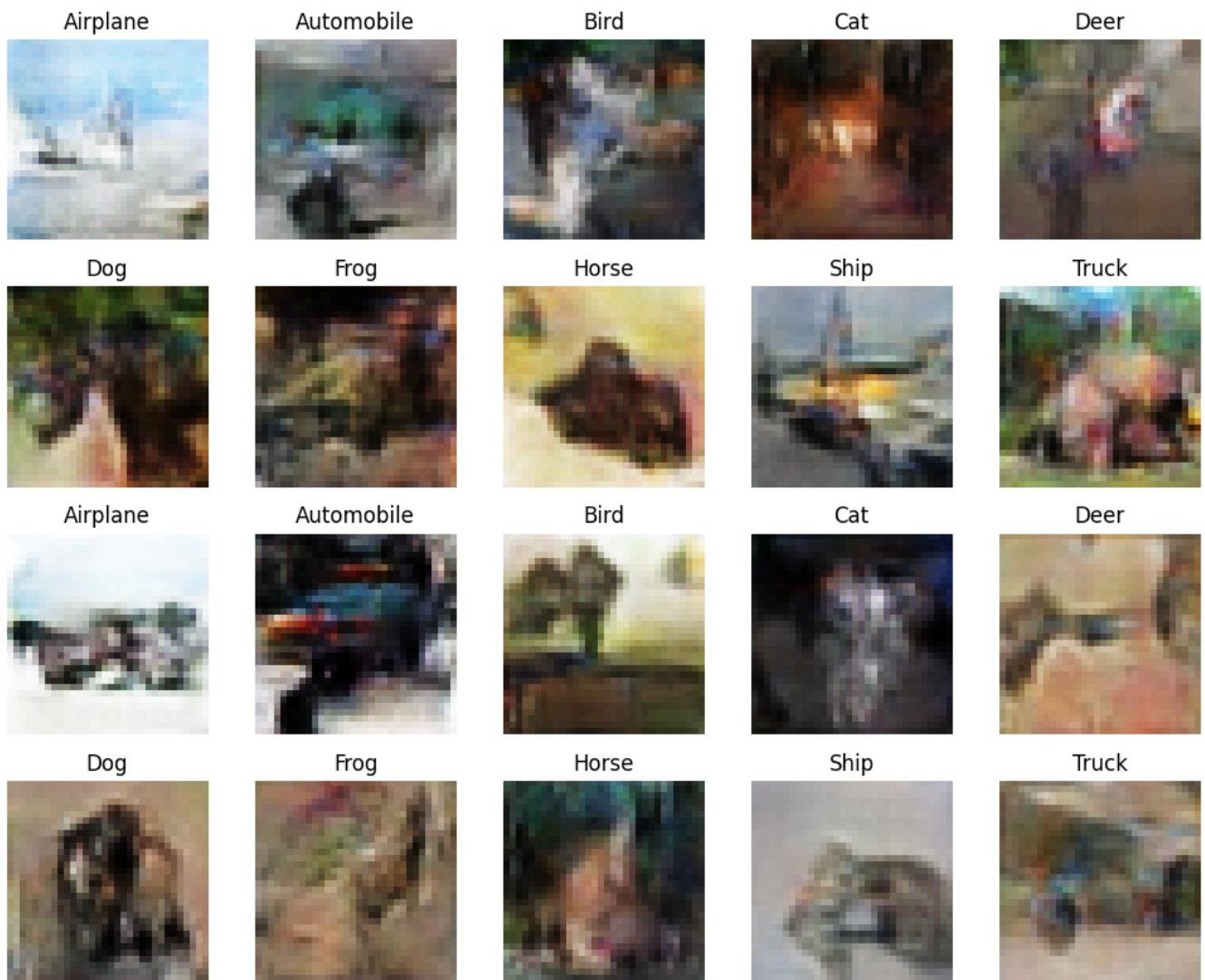
Epoch 19/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2333 - g_loss: 3.1174 - D(x|y): 0.5012 - D(G(z|y)): 0.1266 - KL Divergence: 4.5975

Epoch 20/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2221 - g_loss: 3.1965 - D(x|y): 0.5002 - D(G(z|y)): 0.1241 - KL Divergence: 4.6999

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - ACGAN/generator-epoch-20.h5

Epoch 21/200

782/782 [=====] - 67s 86ms/step - d_loss: 0.2181 - g_loss: 3.3012 - D(x|y): 0.4998 - D(G(z|y)): 0.1191 - KL Divergence: 4.6729

Epoch 22/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.2078 - g_loss: 3.4404 - D(x|y): 0.5006 - D(G(z|y)): 0.1139 - KL Divergence: 4.6279

Epoch 23/200

782/782 [=====] - 67s 85ms/step - d_loss: 0.1955 - g_loss: 3.5669 - D(x|y): 0.5002 - D(G(z|y)): 0.1072 - KL Divergence: 4.6501

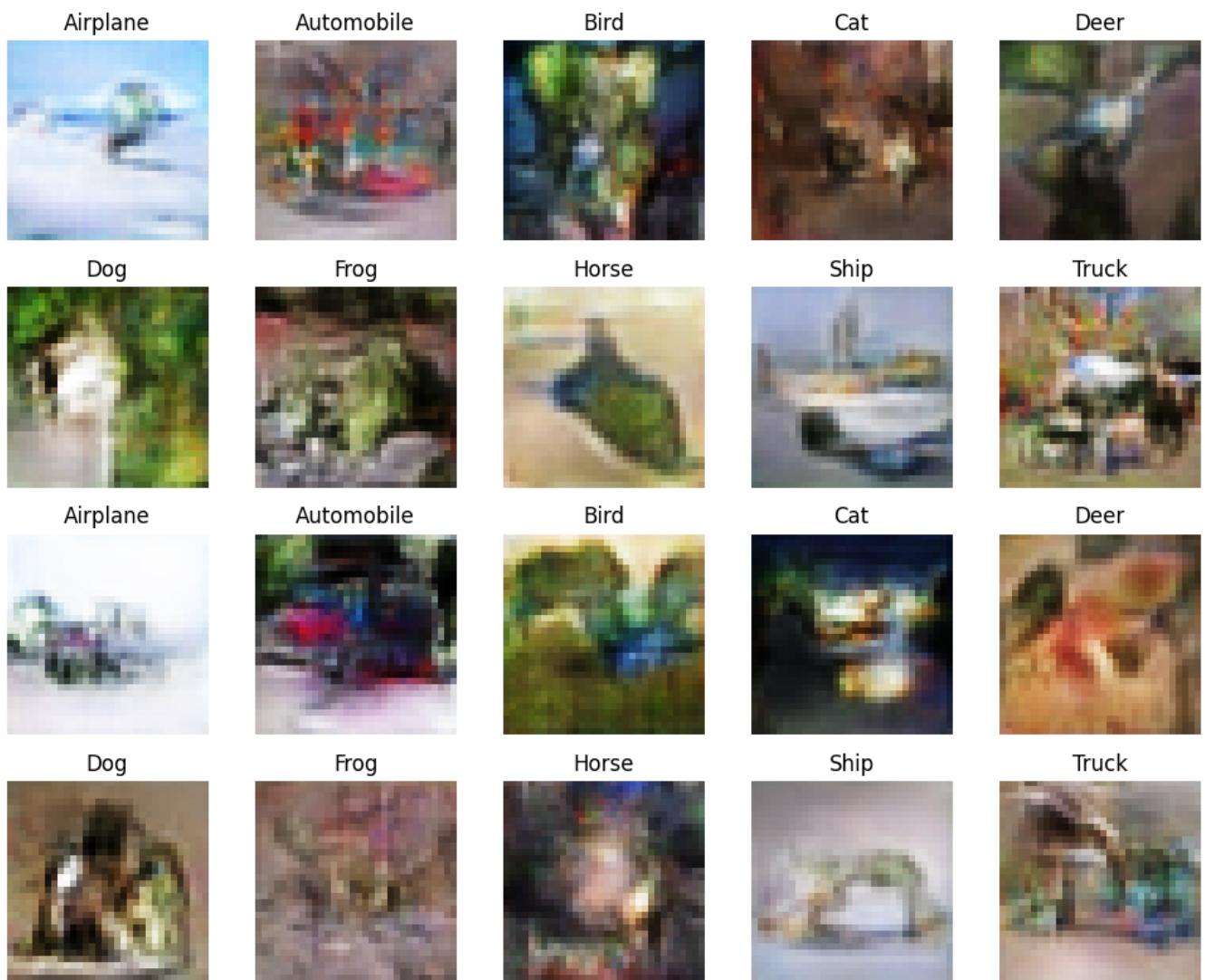
Epoch 24/200

782/782 [=====] - 66s 84ms/step - d_loss: 0.1867 - g_loss: 3.7490 - D(x|y): 0.5002 - D(G(z|y)): 0.1027 - KL Divergence: 4.5787

Epoch 25/200

782/782 [=====] - 71s 91ms/step - d_loss: 0.1894 - g_loss: 3.8604 - D(x|y): 0.5000 - D(G(z|y)): 0.1006 - KL Divergence: 4.5637

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-25.h5

Epoch 26/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1712 - g_loss: 3.9667 - D(x|y): 0.5002 - D(G(z|y)): 0.0960 - KL Divergence: 4.6433

Epoch 27/200

782/782 [=====] - 69s 89ms/step - d_loss: 0.1700 - g_loss: 3.9484 - D(x|y): 0.5003 - D(G(z|y)): 0.0964 - KL Divergence: 4.5281

Epoch 28/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1599 - g_loss: 4.2800 - D(x|y): 0.5005 - D(G(z|y)): 0.0875 - KL Divergence: 4.6995

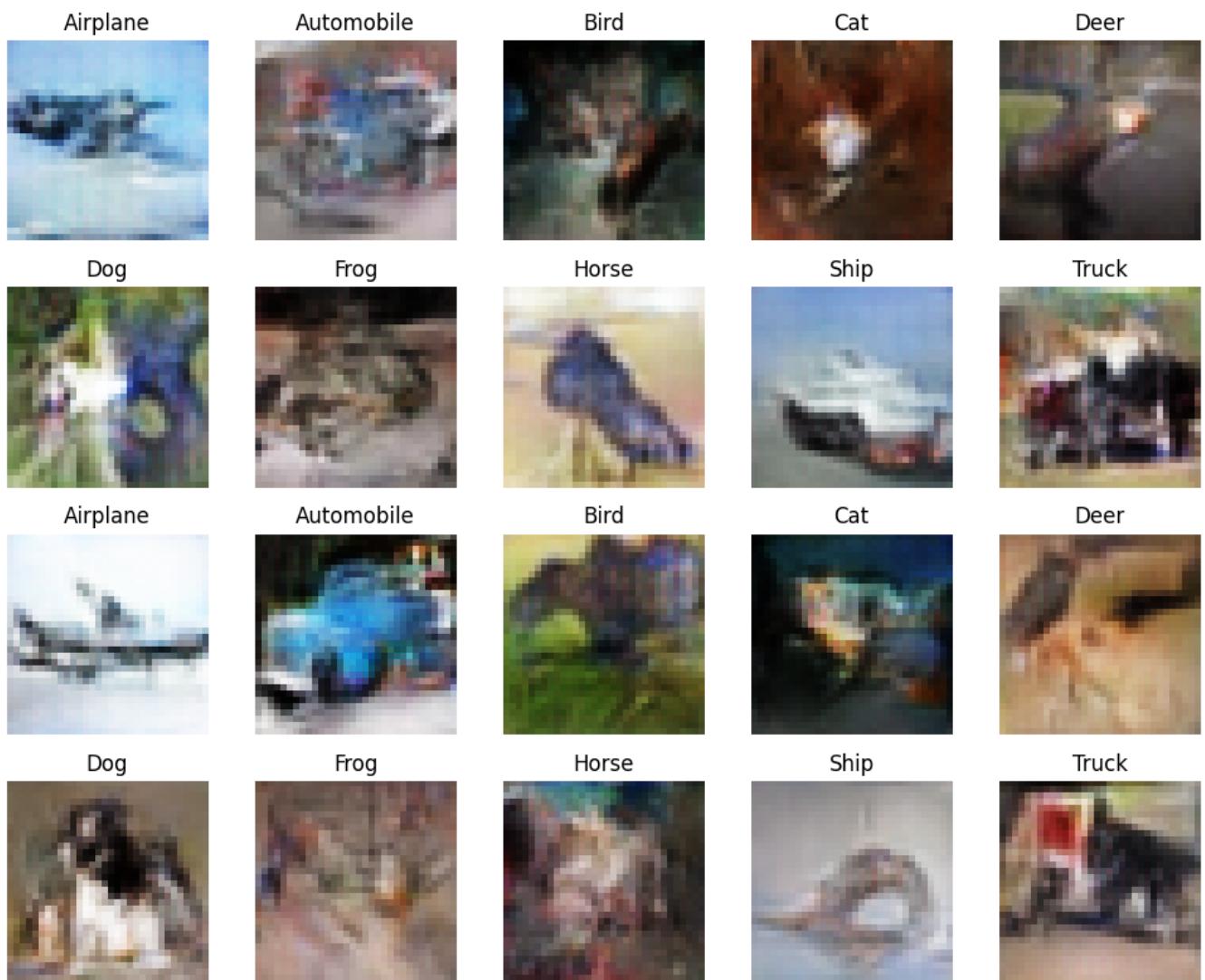
Epoch 29/200

782/782 [=====] - 65s 84ms/step - d_loss: 0.1652 - g_loss: 4.3234 - D(x|y): 0.5003 - D(G(z|y)): 0.0898 - KL Divergence: 4.6046

Epoch 30/200

782/782 [=====] - 74s 94ms/step - d_loss: 0.1529 - g_loss: 4.4116 - D(x|y): 0.4997 - D(G(z|y)): 0.0877 - KL Divergence: 4.5896

1/1 [=====] - 0s 29ms/step



Generator Checkpoint - ACGAN/generator-epoch-30.h5

Epoch 31/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1475 - g_loss: 4.6481 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0825 - KL Divergence: 4.7073

Epoch 32/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1475 - g_loss: 4.6351 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0820 - KL Divergence: 4.7518

Epoch 33/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1445 - g_loss: 4.7635 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0804 - KL Divergence: 4.5958

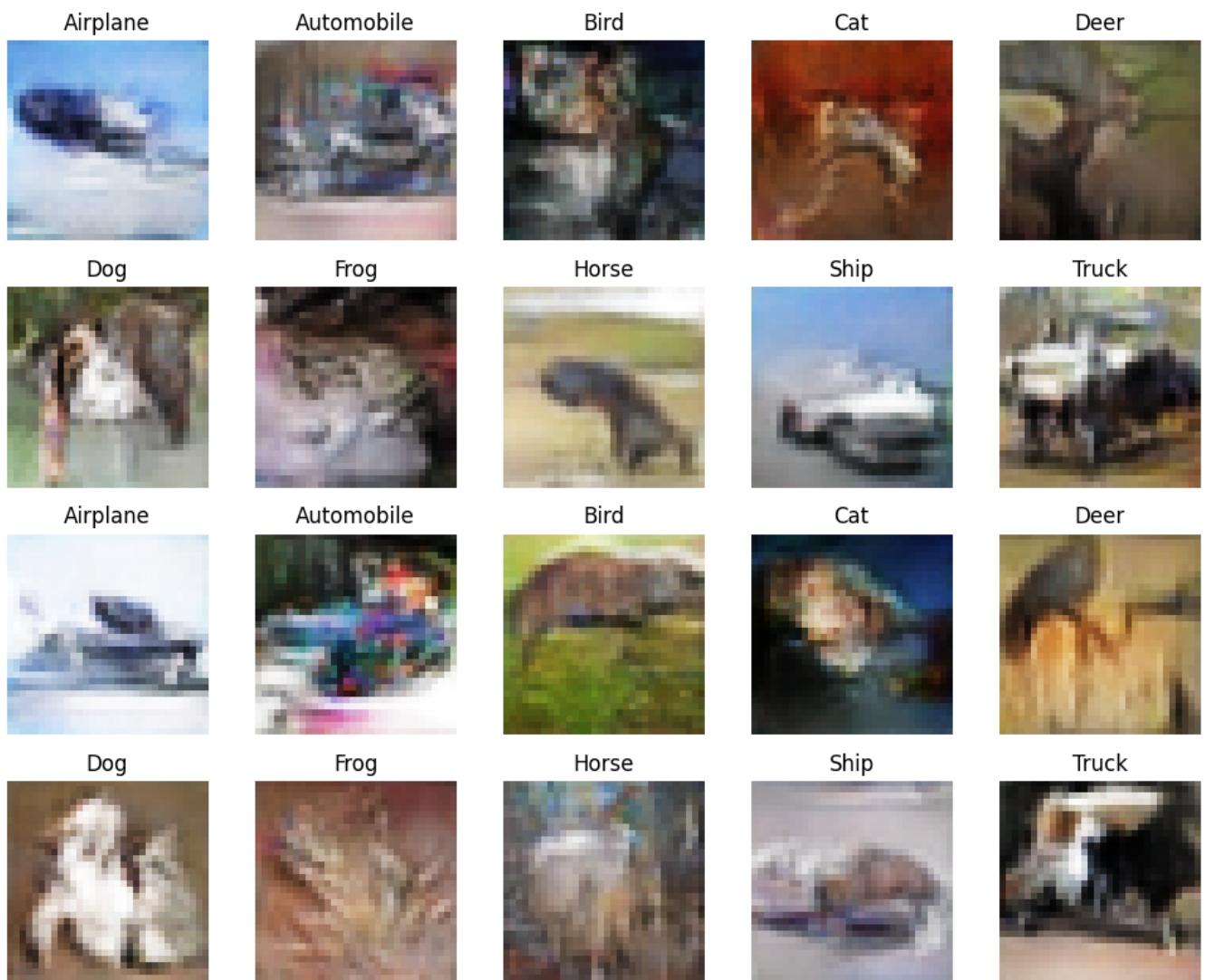
Epoch 34/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1425 - g_loss: 4.8635 -
 $D(x|y)$: 0.5009 - $D(G(z|y))$: 0.0792 - KL Divergence: 4.5395

Epoch 35/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.1403 - g_loss: 5.0478 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.0743 - KL Divergence: 4.5070

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - ACGAN/generator-epoch-35.h5

Epoch 36/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1434 - g_loss: 5.0324 - D(x|y): 0.5002 - D(G(z|y)): 0.0781 - KL Divergence: 4.6303

Epoch 37/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1274 - g_loss: 4.9758 - D(x|y): 0.4992 - D(G(z|y)): 0.0742 - KL Divergence: 4.5329

Epoch 38/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1343 - g_loss: 5.1952 - D(x|y): 0.5005 - D(G(z|y)): 0.0743 - KL Divergence: 4.6429

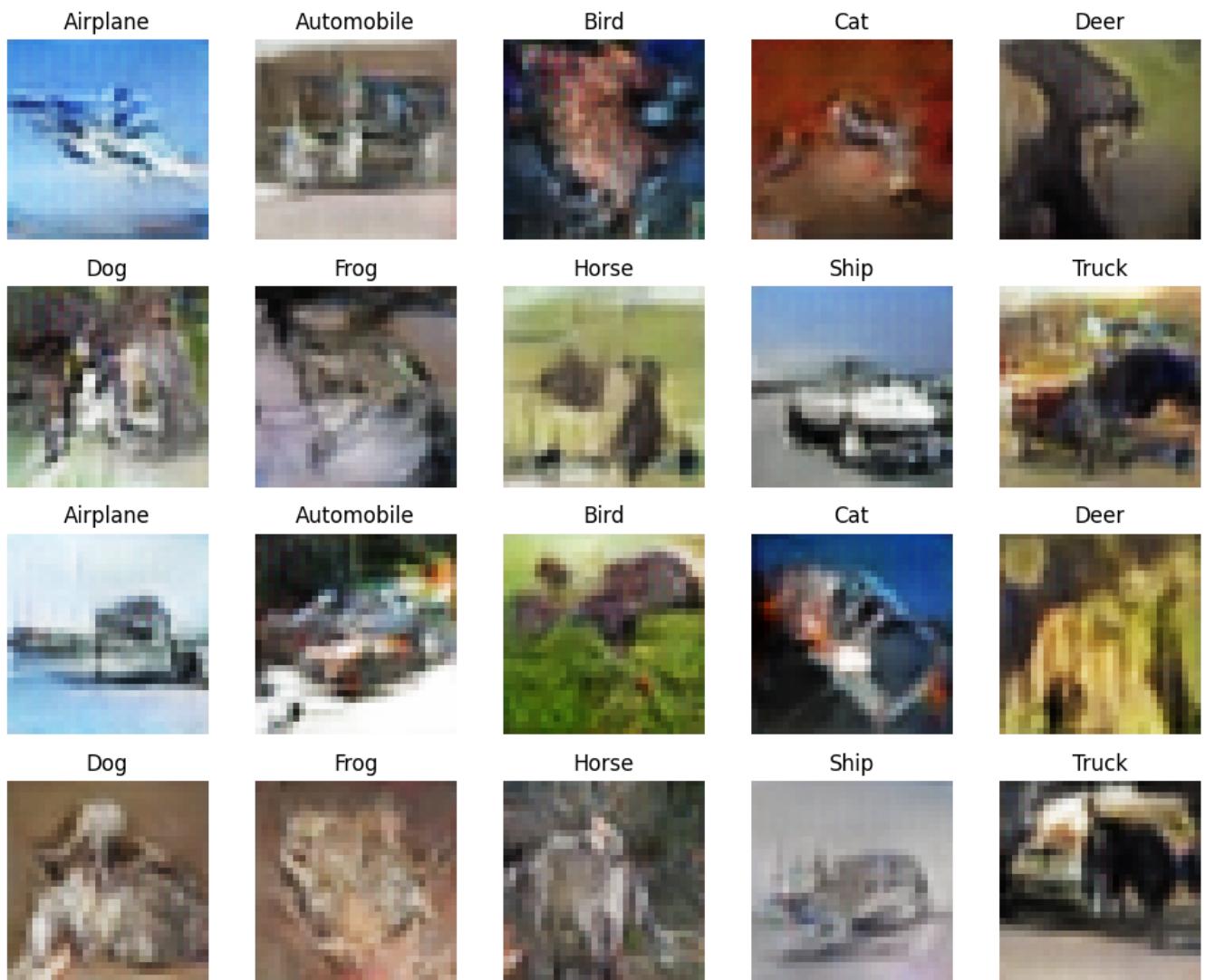
Epoch 39/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.1223 - g_loss: 5.4694 - D(x|y): 0.5002 - D(G(z|y)): 0.0668 - KL Divergence: 4.7246

Epoch 40/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.1275 - g_loss: 5.5361 - D(x|y): 0.5000 - D(G(z|y)): 0.0675 - KL Divergence: 4.5661

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - ACGAN/generator-epoch-40.h5

Epoch 41/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1195 - g_loss: 5.6017 - D(x|y): 0.5003 - D(G(z|y)): 0.0661 - KL Divergence: 4.6437

Epoch 42/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1132 - g_loss: 5.6969 - D(x|y): 0.5001 - D(G(z|y)): 0.0646 - KL Divergence: 4.5650

Epoch 43/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1162 - g_loss: 5.8343 - D(x|y): 0.5001 - D(G(z|y)): 0.0626 - KL Divergence: 4.4984

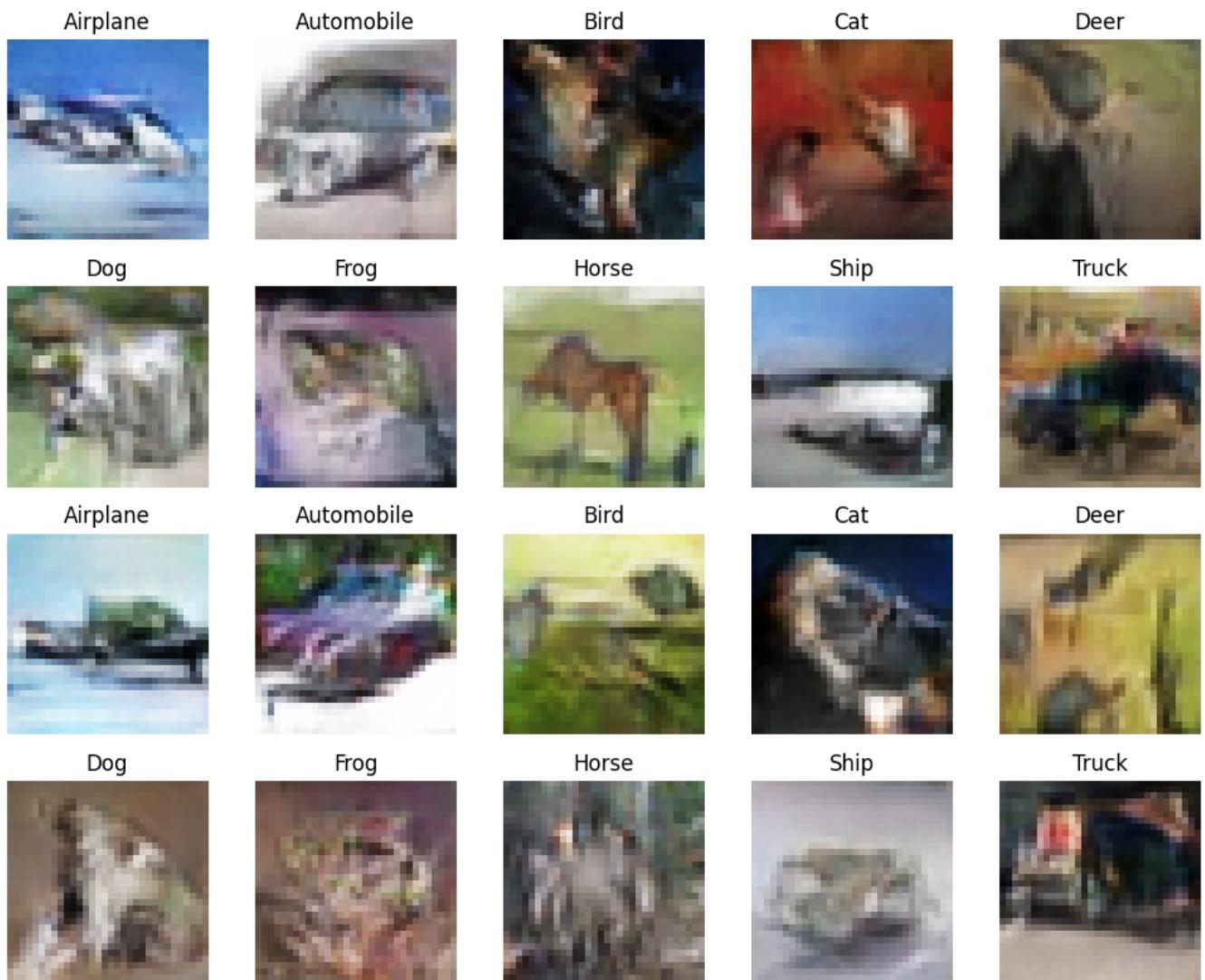
Epoch 44/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.1134 - g_loss: 5.8918 - D(x|y): 0.5001 - D(G(z|y)): 0.0618 - KL Divergence: 4.6466

Epoch 45/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.1095 - g_loss: 5.9400 - D(x|y): 0.5003 - D(G(z|y)): 0.0615 - KL Divergence: 4.6963

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-45.h5

Epoch 46/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1085 - g_loss: 5.9959 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0598$ - KL Divergence: 4.6330

Epoch 47/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1097 - g_loss: 6.1969 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0600$ - KL Divergence: 4.6365

Epoch 48/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.1056 - g_loss: 6.2659 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0571$ - KL Divergence: 4.5335

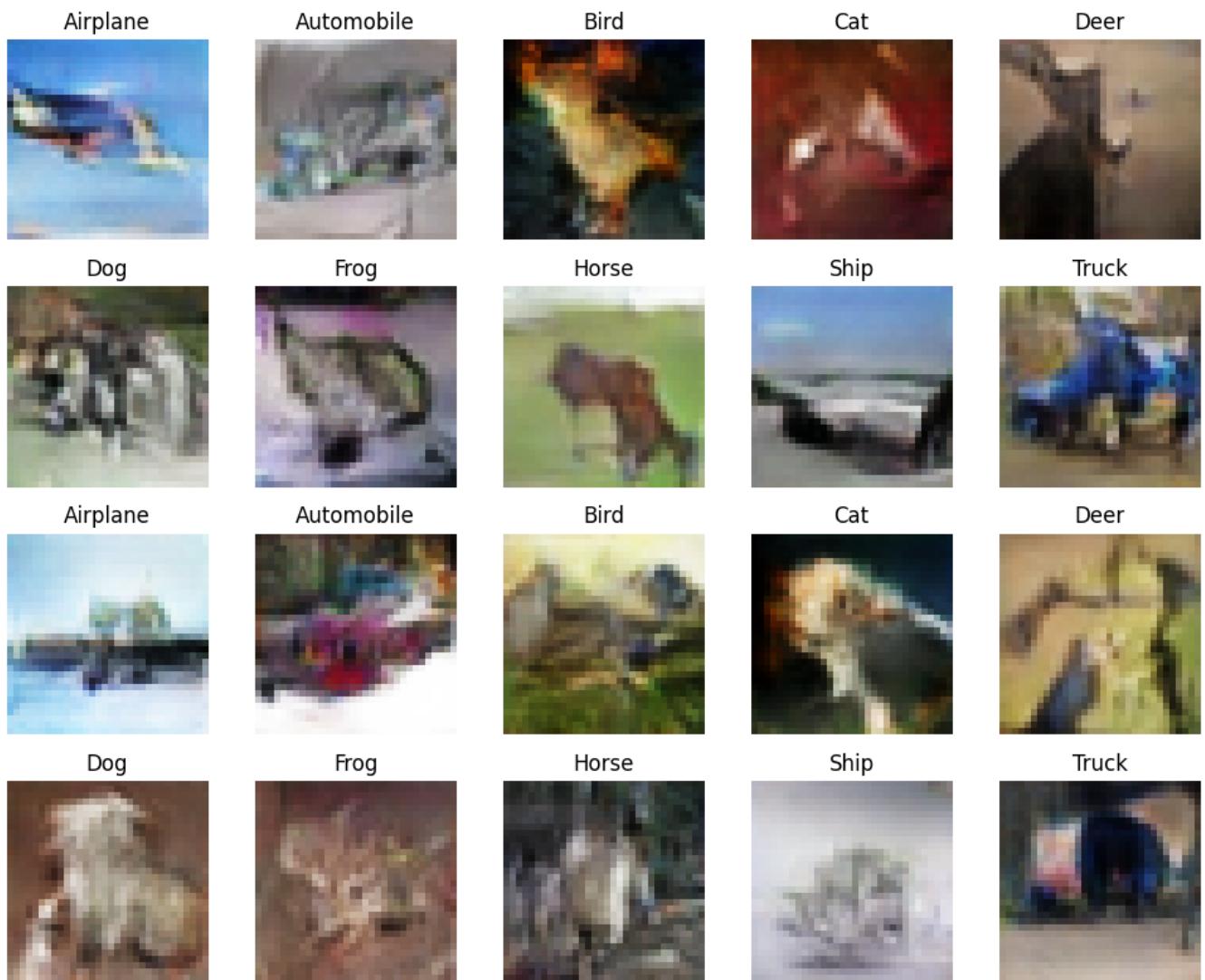
Epoch 49/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.1058 - g_loss: 6.3194 -
 $D(x|y): 0.4998$ - $D(G(z|y)): 0.0575$ - KL Divergence: 4.7878

Epoch 50/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1331 - g_loss: 6.0614 -
 $D(x|y): 0.5018$ - $D(G(z|y)): 0.0646$ - KL Divergence: 4.6140

1/1 [=====] - 0s 28ms/step



Generator Checkpoint - ACGAN/generator-epoch-50.h5

Epoch 51/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0960 - g_loss: 6.4727 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0542$ - KL Divergence: 4.6804

Epoch 52/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0977 - g_loss: 6.5551 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.0528$ - KL Divergence: 4.6576

Epoch 53/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0937 - g_loss: 6.6376 -
 $D(x|y): 0.5005$ - $D(G(z|y)): 0.0509$ - KL Divergence: 4.5724

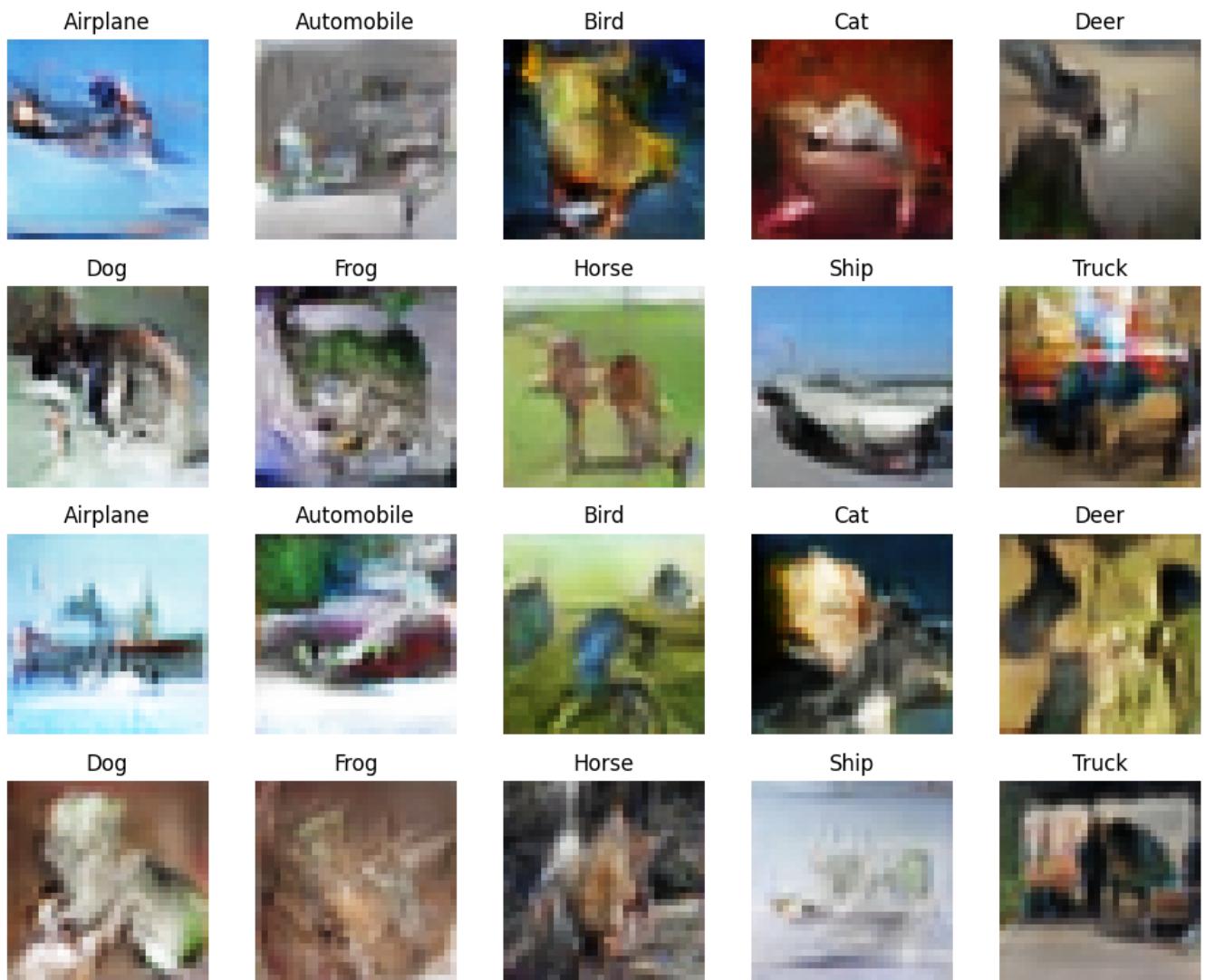
Epoch 54/200

782/782 [=====] - 65s 82ms/step - d_loss: 0.0917 - g_loss: 6.8088 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0499$ - KL Divergence: 4.5355

Epoch 55/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1422 - g_loss: 6.1903 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.0664$ - KL Divergence: 4.5857

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - ACGAN/generator-epoch-55.h5

Epoch 56/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0917 - g_loss: 6.6402 - D(x|y): 0.5000 - D(G(z|y)): 0.0522 - KL Divergence: 4.5267

Epoch 57/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0883 - g_loss: 6.7616 - D(x|y): 0.5001 - D(G(z|y)): 0.0487 - KL Divergence: 4.6800

Epoch 58/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0893 - g_loss: 6.9044 - D(x|y): 0.5003 - D(G(z|y)): 0.0515 - KL Divergence: 4.7052

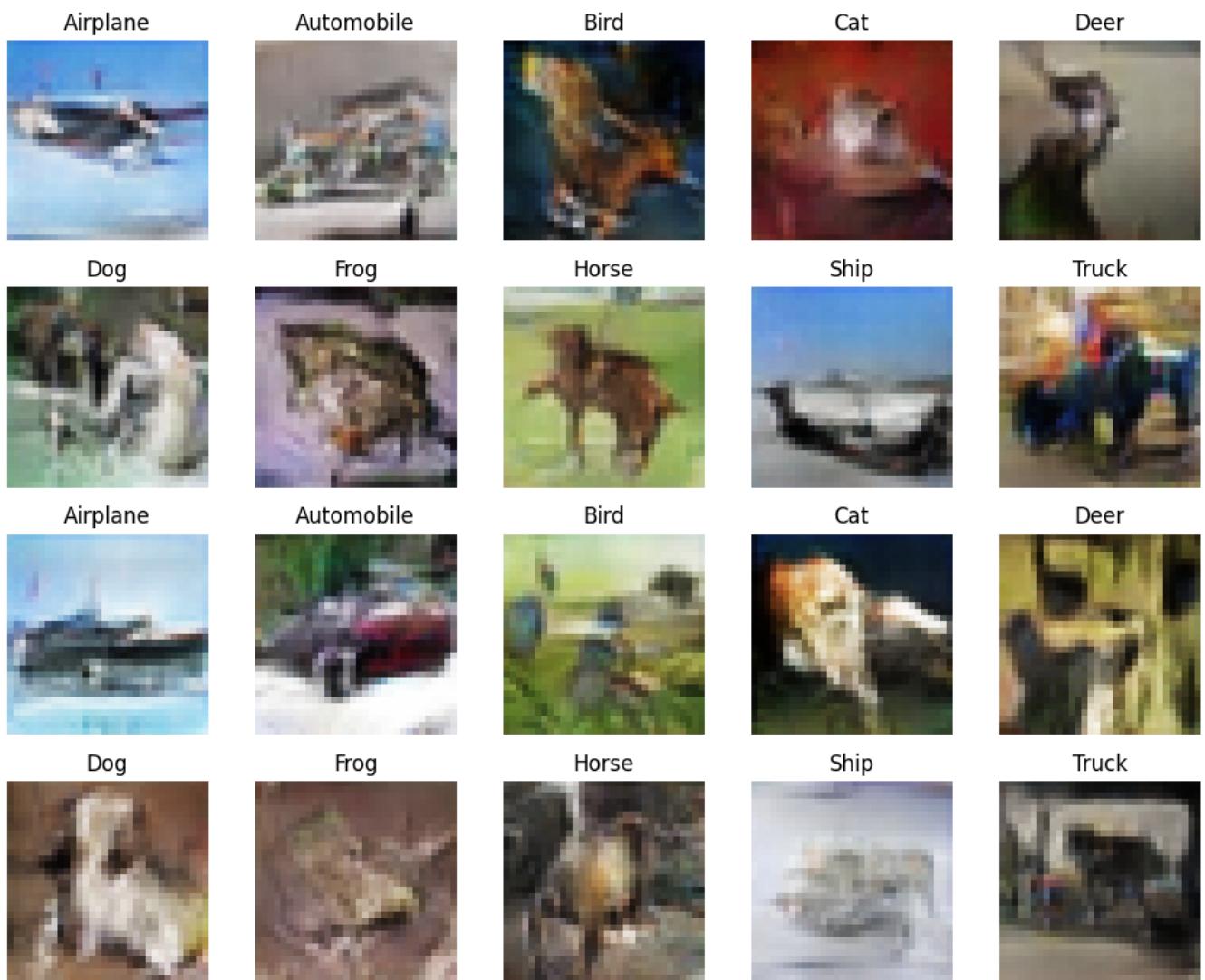
Epoch 59/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0901 - g_loss: 6.9616 - D(x|y): 0.5001 - D(G(z|y)): 0.0493 - KL Divergence: 4.6182

Epoch 60/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.1194 - g_loss: 6.7140 - D(x|y): 0.5009 - D(G(z|y)): 0.0552 - KL Divergence: 4.6978

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - ACGAN/generator-epoch-60.h5

Epoch 61/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0835 - g_loss: 7.2504 - D(x|y): 0.4999 - D(G(z|y)): 0.0439 - KL Divergence: 4.5707

Epoch 62/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0819 - g_loss: 7.1562 - D(x|y): 0.4999 - D(G(z|y)): 0.0435 - KL Divergence: 4.4694

Epoch 63/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0855 - g_loss: 7.2073 - D(x|y): 0.5000 - D(G(z|y)): 0.0474 - KL Divergence: 4.6348

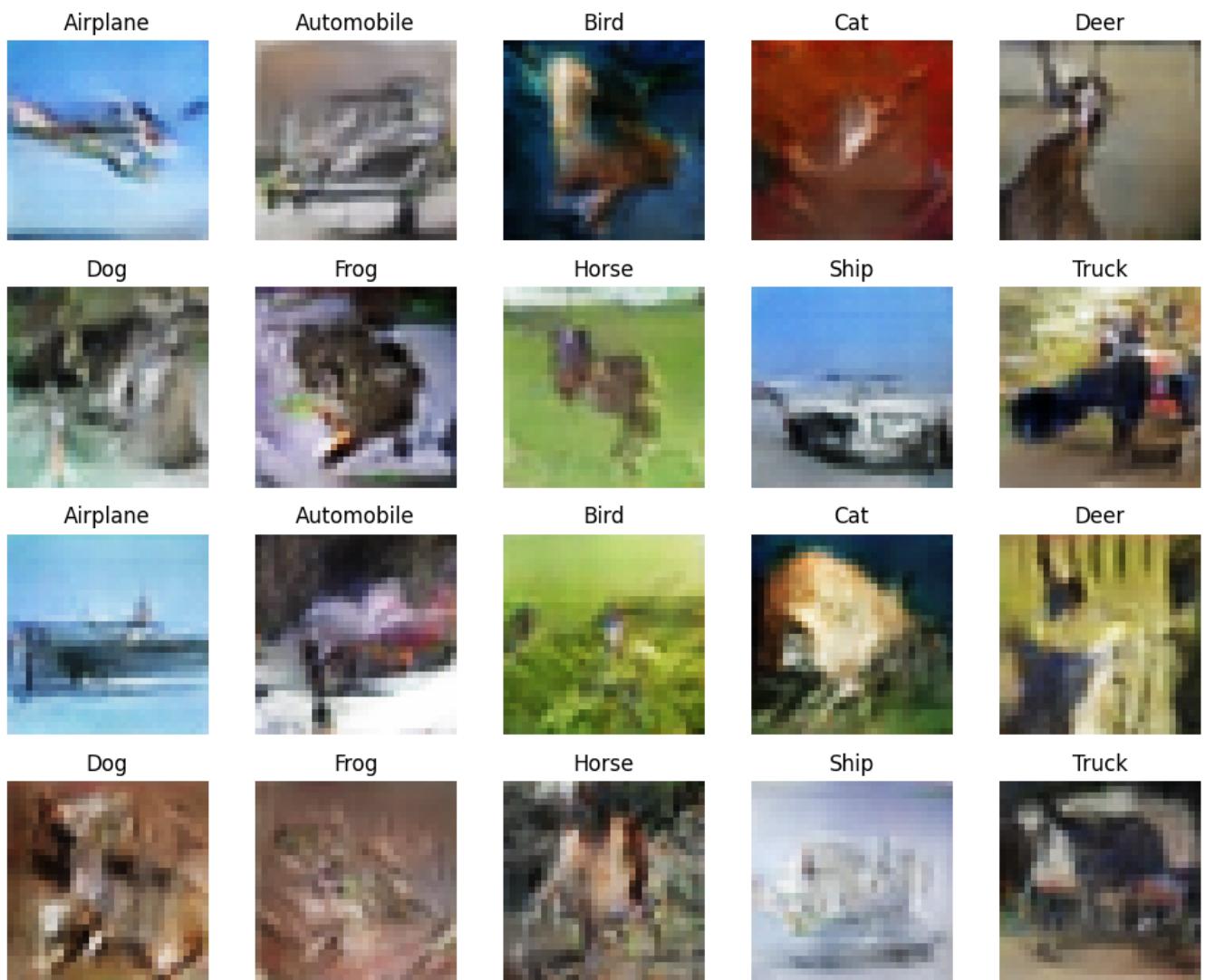
Epoch 64/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0849 - g_loss: 7.3213 - D(x|y): 0.5000 - D(G(z|y)): 0.0448 - KL Divergence: 4.5846

Epoch 65/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0876 - g_loss: 7.2725 - D(x|y): 0.4997 - D(G(z|y)): 0.0459 - KL Divergence: 4.5221

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-65.h5

Epoch 66/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0805 - g_loss: 7.5003 - D(x|y): 0.4998 - D(G(z|y)): 0.0416 - KL Divergence: 4.6065

Epoch 67/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0825 - g_loss: 7.4254 - D(x|y): 0.4999 - D(G(z|y)): 0.0443 - KL Divergence: 4.5317

Epoch 68/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0809 - g_loss: 7.5338 - D(x|y): 0.5003 - D(G(z|y)): 0.0418 - KL Divergence: 4.5732

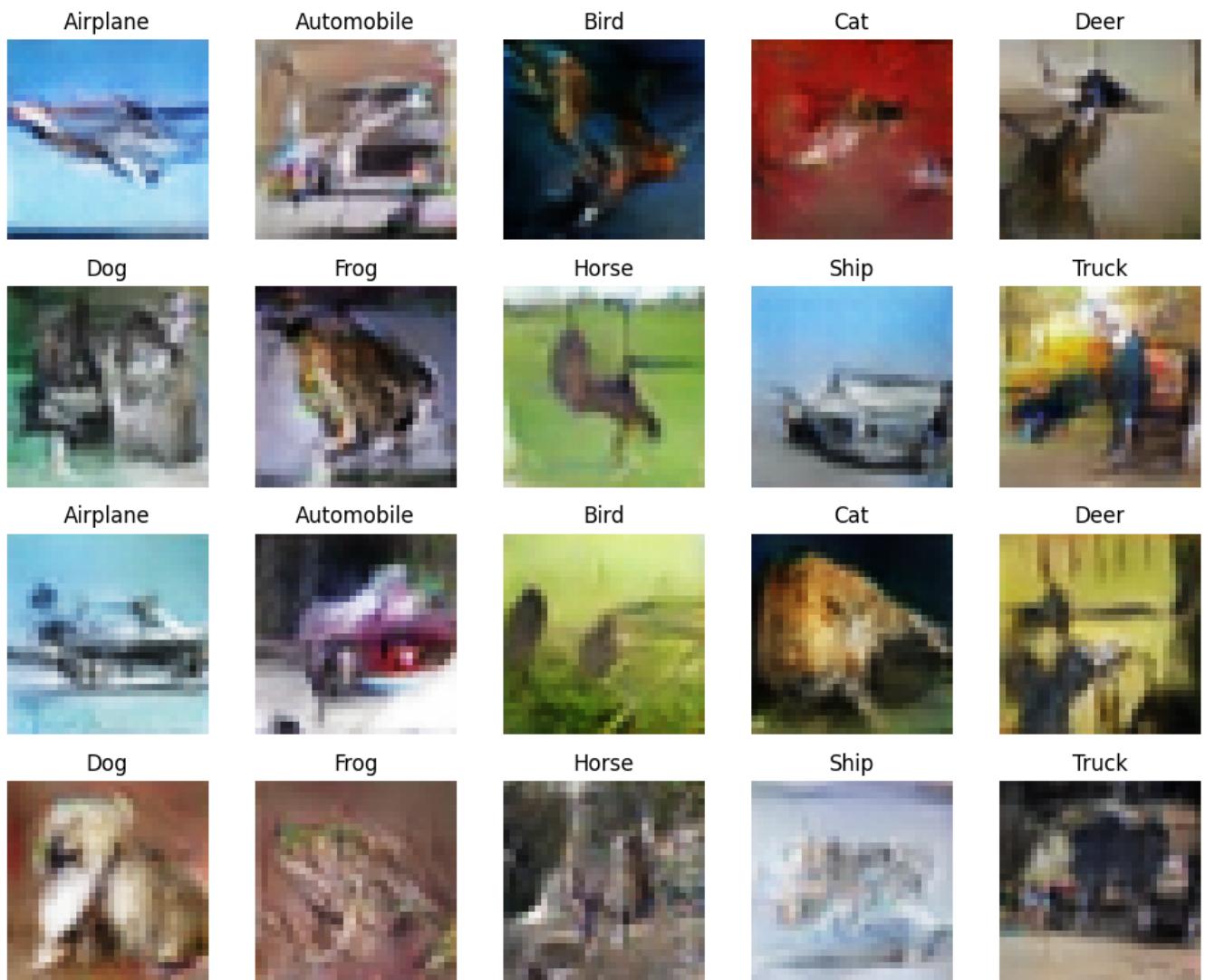
Epoch 69/200

782/782 [=====] - 65s 82ms/step - d_loss: 0.0777 - g_loss: 7.5679 - D(x|y): 0.5004 - D(G(z|y)): 0.0426 - KL Divergence: 4.6603

Epoch 70/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0779 - g_loss: 7.6933 - D(x|y): 0.5002 - D(G(z|y)): 0.0406 - KL Divergence: 4.5582

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-70.h5

Epoch 71/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0770 - g_loss: 7.6865 - D(x|y): 0.5004 - D(G(z|y)): 0.0418 - KL Divergence: 4.5858

Epoch 72/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0761 - g_loss: 7.7936 - D(x|y): 0.5000 - D(G(z|y)): 0.0404 - KL Divergence: 4.6475

Epoch 73/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0745 - g_loss: 7.7917 - D(x|y): 0.5003 - D(G(z|y)): 0.0416 - KL Divergence: 4.8225

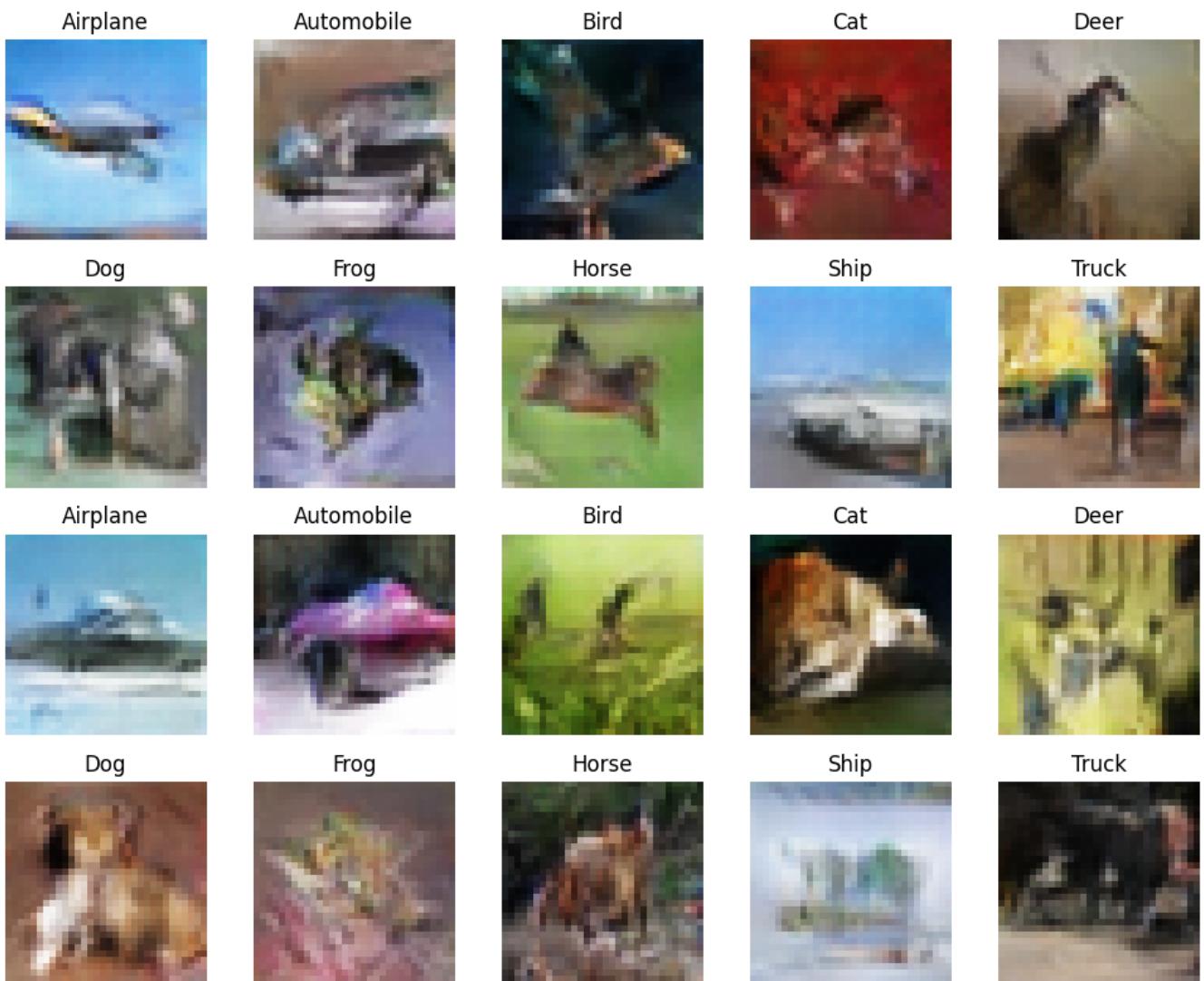
Epoch 74/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0780 - g_loss: 7.9089 - D(x|y): 0.5000 - D(G(z|y)): 0.0426 - KL Divergence: 4.6795

Epoch 75/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0726 - g_loss: 7.9694 - D(x|y): 0.5003 - D(G(z|y)): 0.0401 - KL Divergence: 4.5963

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - ACGAN/generator-epoch-75.h5

Epoch 76/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0740 - g_loss: 8.0976 - D(x|y): 0.4998 - D(G(z|y)): 0.0357 - KL Divergence: 4.6527

Epoch 77/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0689 - g_loss: 8.1082 - D(x|y): 0.5002 - D(G(z|y)): 0.0373 - KL Divergence: 4.5845

Epoch 78/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0714 - g_loss: 8.1449 - D(x|y): 0.5001 - D(G(z|y)): 0.0363 - KL Divergence: 4.5820

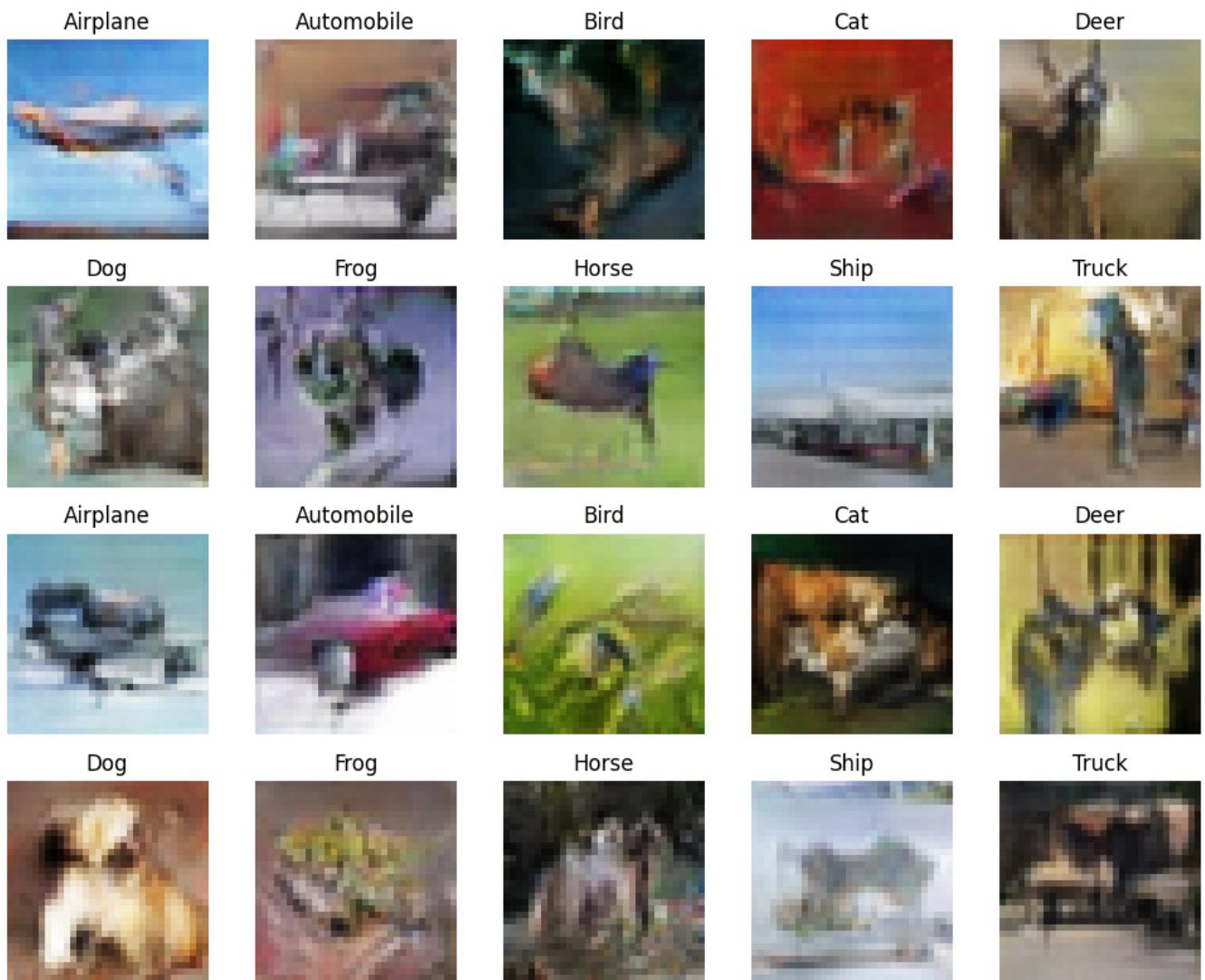
Epoch 79/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0698 - g_loss: 8.0619 - D(x|y): 0.4998 - D(G(z|y)): 0.0382 - KL Divergence: 4.6510

Epoch 80/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0721 - g_loss: 8.1509 - D(x|y): 0.5001 - D(G(z|y)): 0.0355 - KL Divergence: 4.5917

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - ACGAN/generator-epoch-80.h5

Epoch 81/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0711 - g_loss: 8.2037 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0371$ - KL Divergence: 4.5868

Epoch 82/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0699 - g_loss: 8.2055 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0380$ - KL Divergence: 4.5820

Epoch 83/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0638 - g_loss: 8.2898 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0346$ - KL Divergence: 4.6216

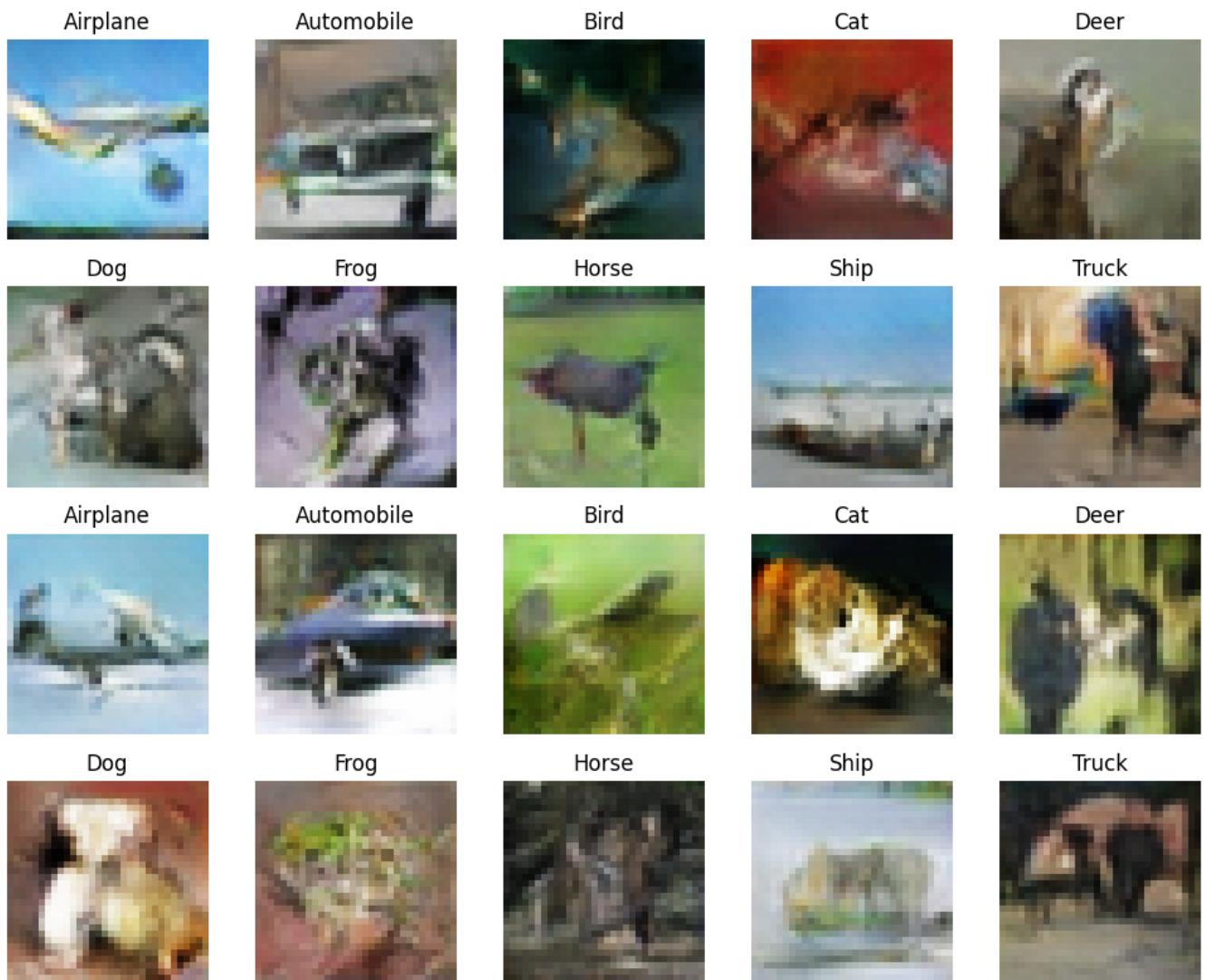
Epoch 84/200

782/782 [=====] - 65s 82ms/step - d_loss: 0.0677 - g_loss: 8.4533 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0352$ - KL Divergence: 4.5663

Epoch 85/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0647 - g_loss: 8.3809 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.0364$ - KL Divergence: 4.5817

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-85.h5

Epoch 86/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0667 - g_loss: 8.3906 - D(x|y): 0.5001 - D(G(z|y)): 0.0352 - KL Divergence: 4.5290

Epoch 87/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0650 - g_loss: 8.4169 - D(x|y): 0.5000 - D(G(z|y)): 0.0361 - KL Divergence: 4.4771

Epoch 88/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0645 - g_loss: 8.4398 - D(x|y): 0.5000 - D(G(z|y)): 0.0341 - KL Divergence: 4.4817

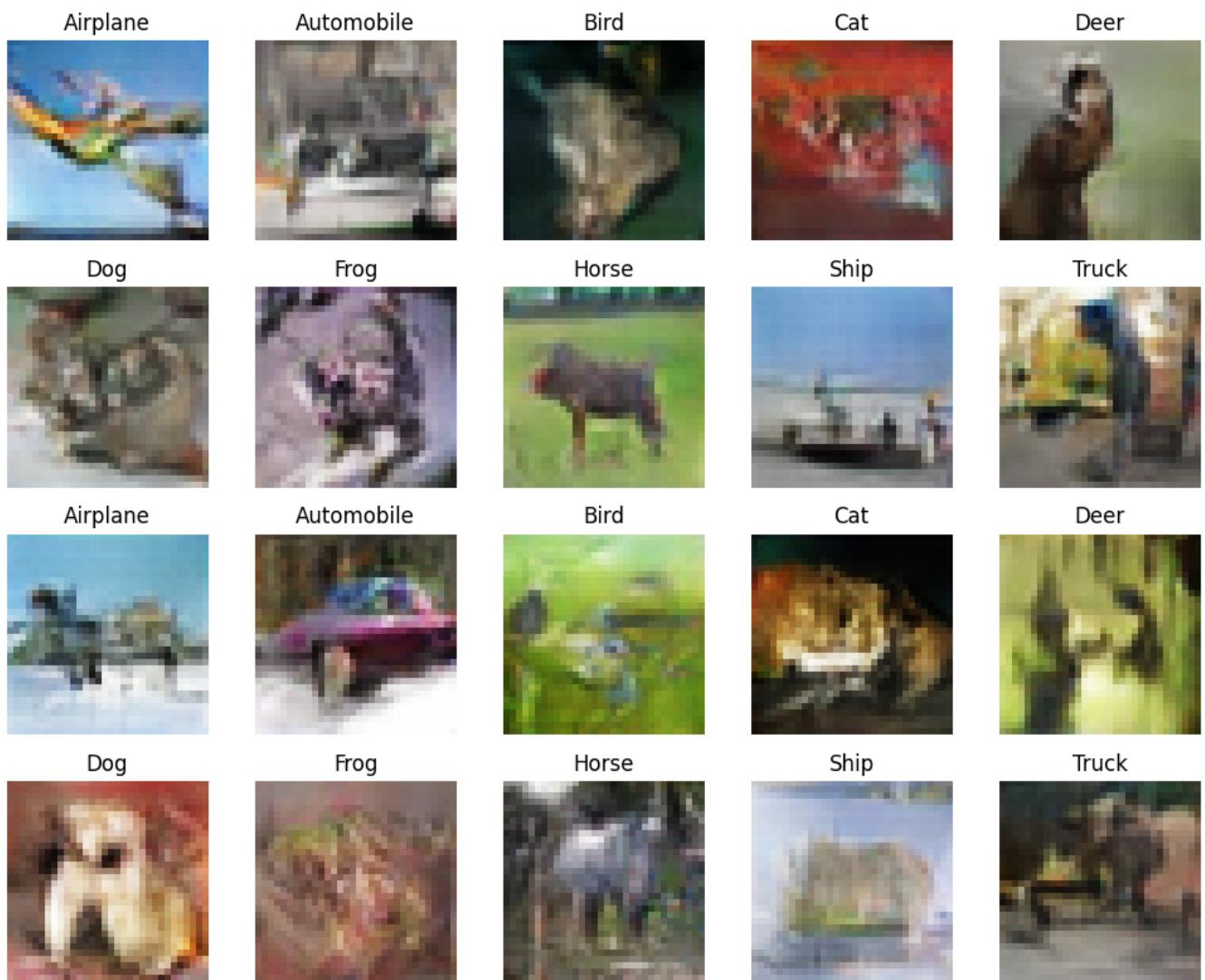
Epoch 89/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0614 - g_loss: 8.5592 - D(x|y): 0.5001 - D(G(z|y)): 0.0333 - KL Divergence: 4.5572

Epoch 90/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0607 - g_loss: 8.6394 - D(x|y): 0.5001 - D(G(z|y)): 0.0340 - KL Divergence: 4.4775

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - ACGAN/generator-epoch-90.h5

Epoch 91/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0609 - g_loss: 8.6527 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0328 - KL Divergence: 4.4637

Epoch 92/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0624 - g_loss: 8.6129 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0321 - KL Divergence: 4.4784

Epoch 93/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0644 - g_loss: 8.6866 -
 $D(x|y)$: 0.4996 - $D(G(z|y))$: 0.0336 - KL Divergence: 4.6336

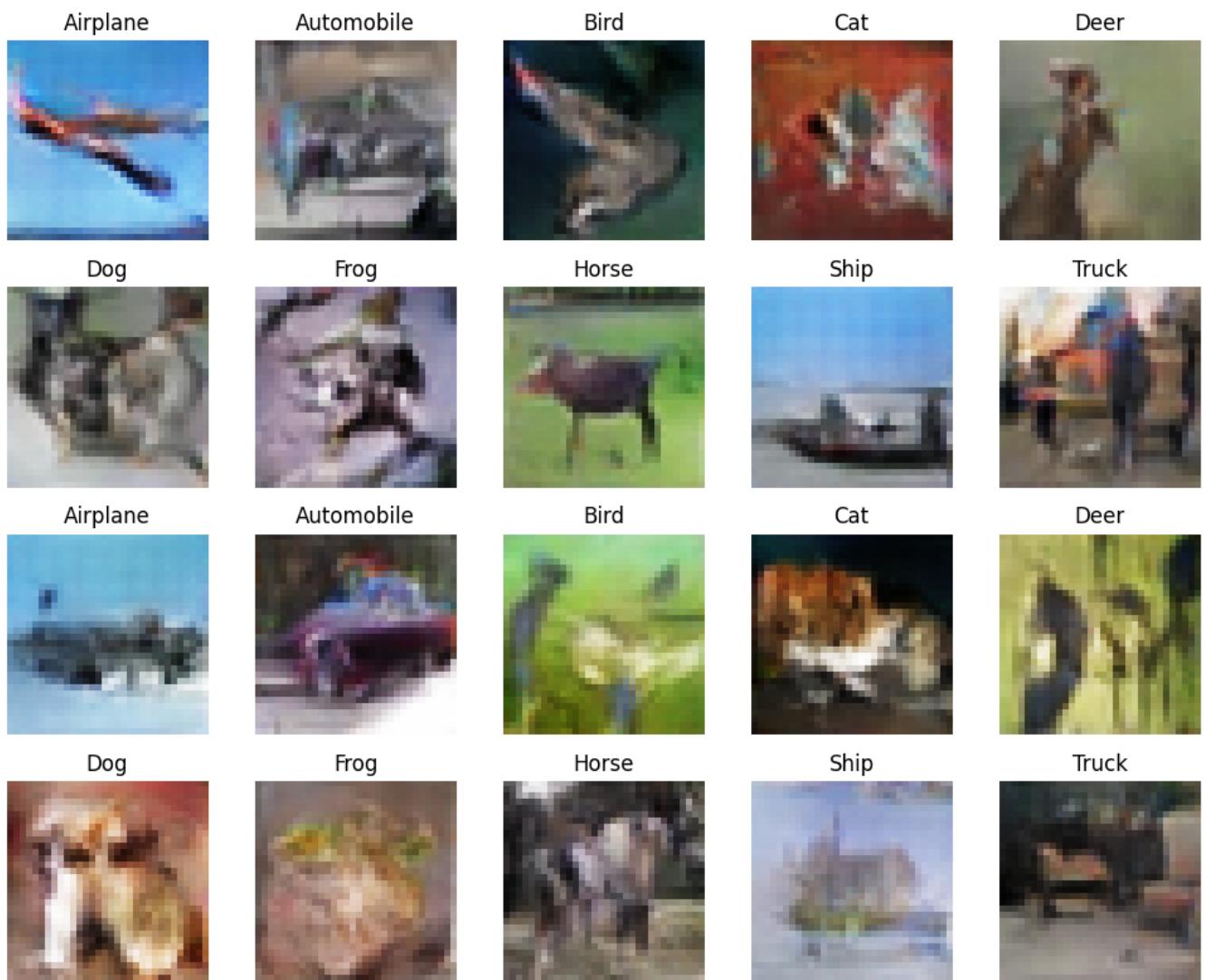
Epoch 94/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0624 - g_loss: 8.7968 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0334 - KL Divergence: 4.5344

Epoch 95/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0613 - g_loss: 8.9031 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0312 - KL Divergence: 4.4683

1/1 [=====] - 0s 22ms/step



Generator Checkpoint - ACGAN/generator-epoch-95.h5

Epoch 96/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0655 - g_loss: 8.7648 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0331 - KL Divergence: 4.4158

Epoch 97/200

782/782 [=====] - 71s 90ms/step - d_loss: 0.0615 - g_loss: 8.7661 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.0324 - KL Divergence: 4.5183

Epoch 98/200

782/782 [=====] - 70s 90ms/step - d_loss: 0.0568 - g_loss: 8.9036 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0303 - KL Divergence: 4.4838

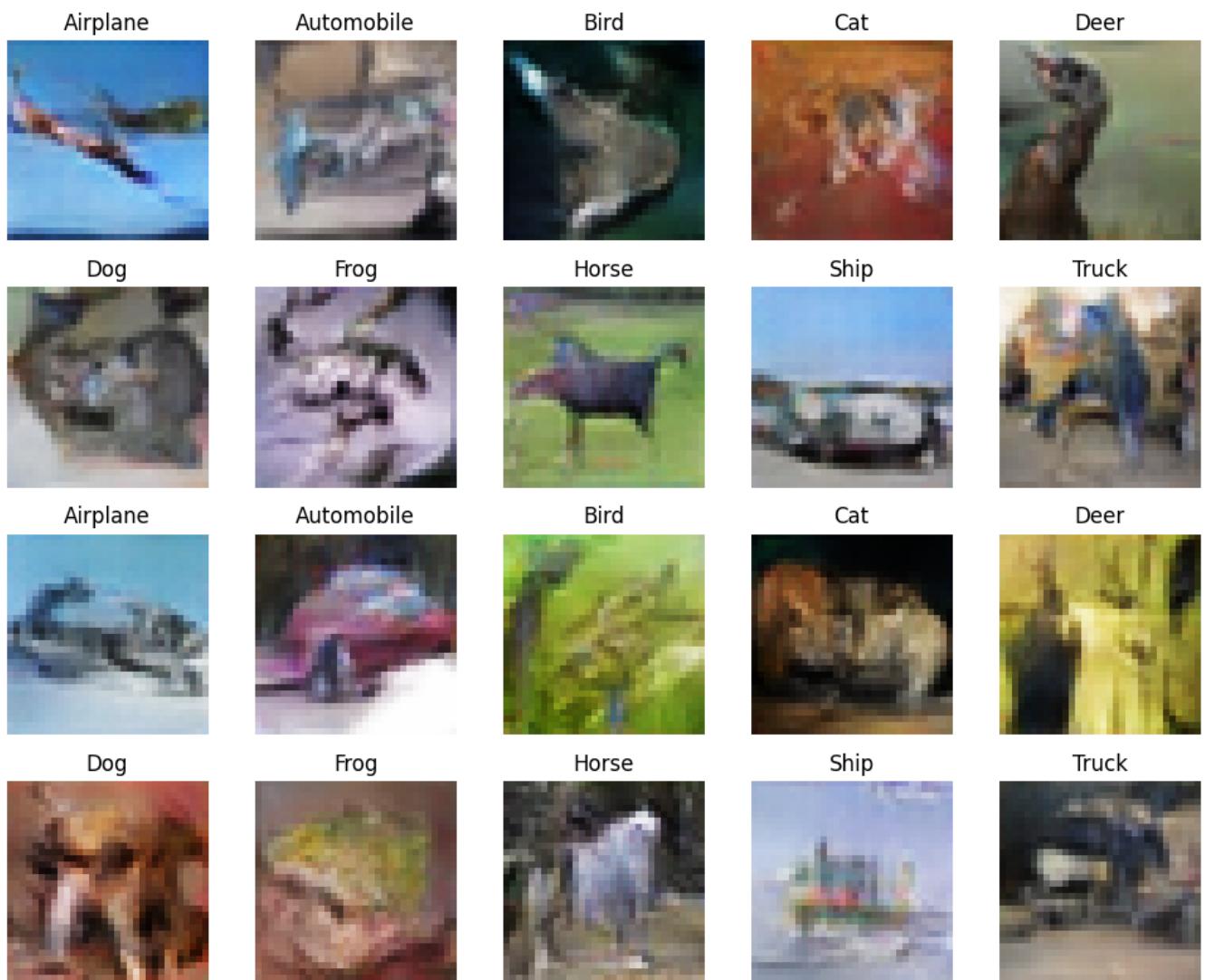
Epoch 99/200

782/782 [=====] - 70s 89ms/step - d_loss: 0.0563 - g_loss: 9.0012 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0301 - KL Divergence: 4.5777

Epoch 100/200

782/782 [=====] - 70s 89ms/step - d_loss: 0.0559 - g_loss: 9.0489 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0299 - KL Divergence: 4.6287

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - ACGAN/generator-epoch-100.h5

Epoch 101/200

782/782 [=====] - 66s 85ms/step - d_loss: 0.0548 - g_loss: 9.0293 - D(x|y): 0.5002 - D(G(z|y)): 0.0296 - KL Divergence: 4.7024

Epoch 102/200

782/782 [=====] - 70s 89ms/step - d_loss: 0.0576 - g_loss: 8.8470 - D(x|y): 0.5000 - D(G(z|y)): 0.0332 - KL Divergence: 4.7036

Epoch 103/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.0530 - g_loss: 9.2536 - D(x|y): 0.5001 - D(G(z|y)): 0.0279 - KL Divergence: 4.6197

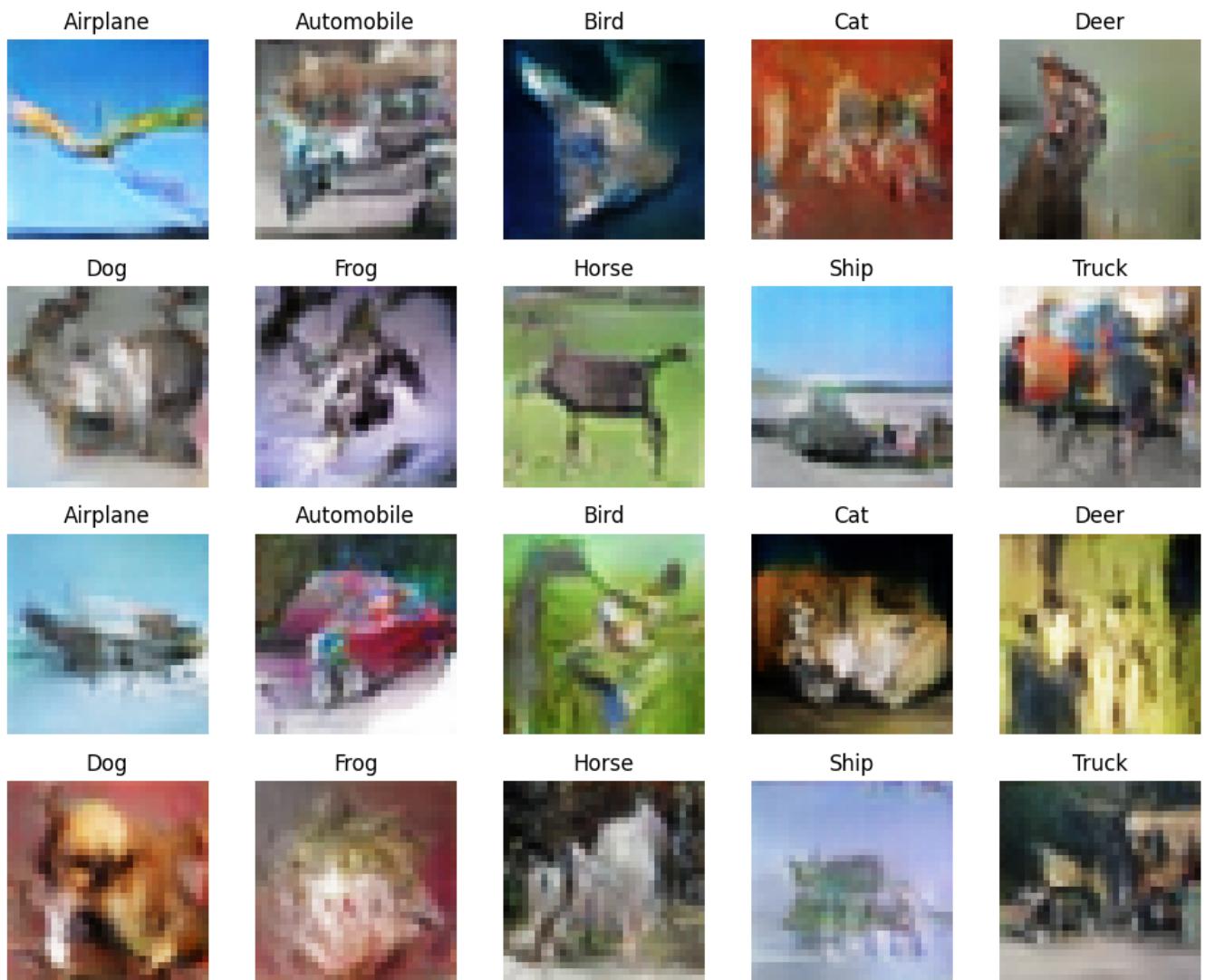
Epoch 104/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0564 - g_loss: 9.0213 - D(x|y): 0.5000 - D(G(z|y)): 0.0311 - KL Divergence: 4.6464

Epoch 105/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0580 - g_loss: 9.1451 - D(x|y): 0.5003 - D(G(z|y)): 0.0304 - KL Divergence: 4.5999

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-105.h5

Epoch 106/200

782/782 [=====] - 65s 82ms/step - d_loss: 0.0535 - g_loss: 9.1489 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0283$ - KL Divergence: 4.5447

Epoch 107/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0540 - g_loss: 9.1239 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0295$ - KL Divergence: 4.4865

Epoch 108/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0548 - g_loss: 9.2245 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0312$ - KL Divergence: 4.5559

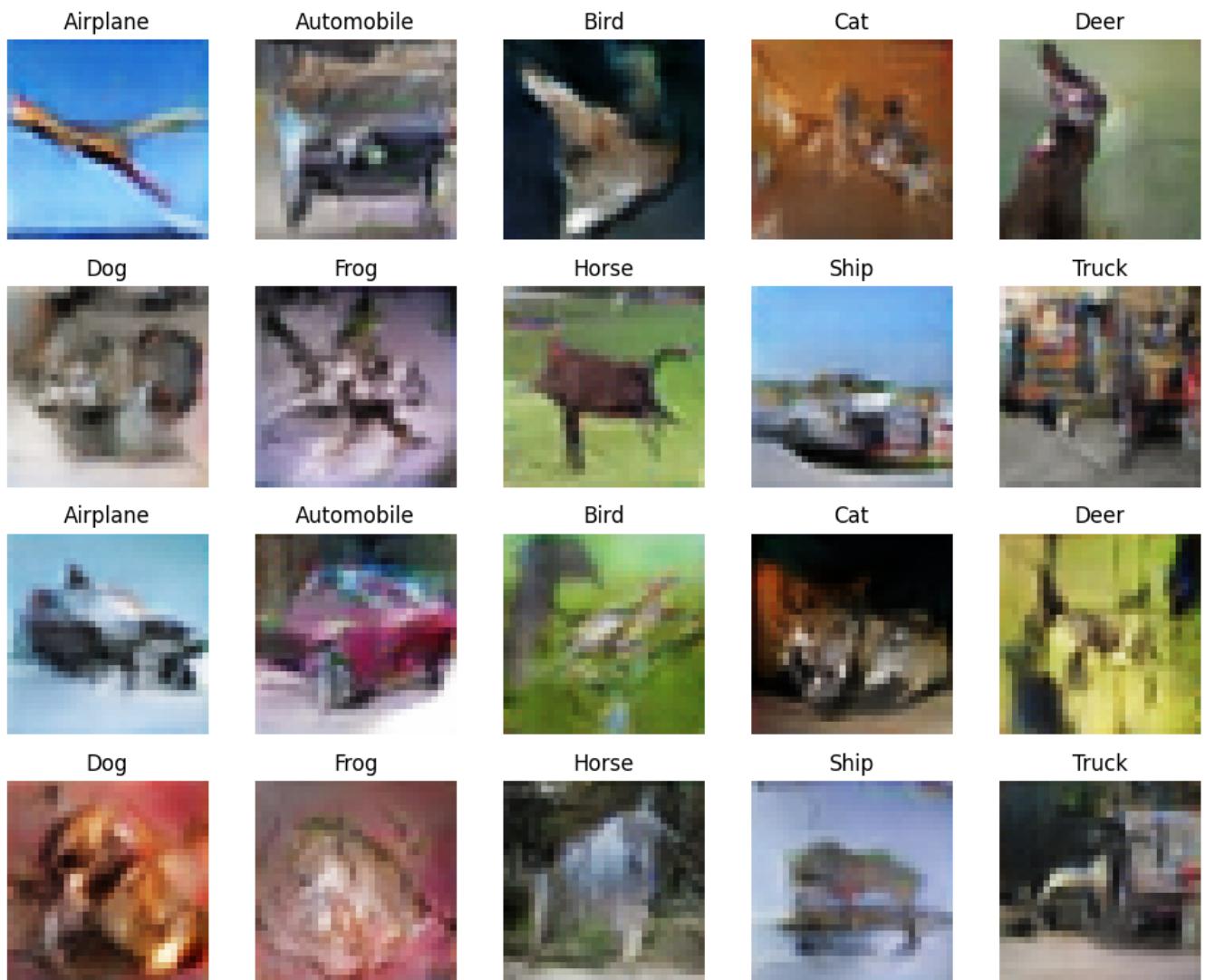
Epoch 109/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0568 - g_loss: 9.1778 -
 $D(x|y): 0.4995$ - $D(G(z|y)): 0.0286$ - KL Divergence: 4.5003

Epoch 110/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0568 - g_loss: 9.2491 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0302$ - KL Divergence: 4.6893

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - ACGAN/generator-epoch-110.h5

Epoch 111/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0567 - g_loss: 9.3953 - D(x|y): 0.5002 - D(G(z|y)): 0.0273 - KL Divergence: 4.6287

Epoch 112/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0528 - g_loss: 9.3565 - D(x|y): 0.5003 - D(G(z|y)): 0.0282 - KL Divergence: 4.7043

Epoch 113/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0526 - g_loss: 9.4539 - D(x|y): 0.5001 - D(G(z|y)): 0.0285 - KL Divergence: 4.5758

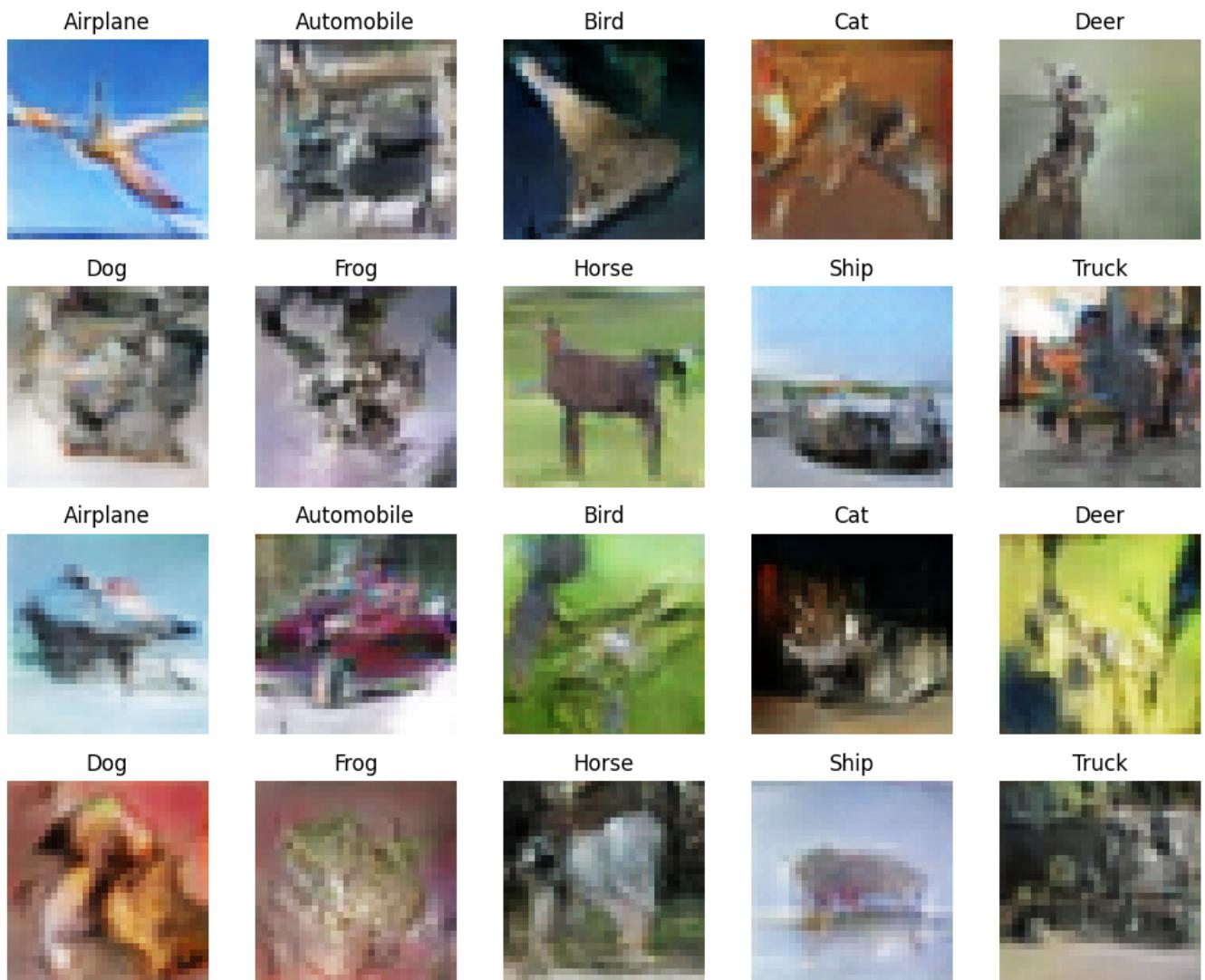
Epoch 114/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0507 - g_loss: 9.5205 - D(x|y): 0.5001 - D(G(z|y)): 0.0257 - KL Divergence: 4.4659

Epoch 115/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0531 - g_loss: 9.5134 - D(x|y): 0.5001 - D(G(z|y)): 0.0271 - KL Divergence: 4.5143

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-115.h5

Epoch 116/200

782/782 [=====] - 65s 82ms/step - d_loss: 0.0511 - g_loss: 9.5440 -
 $D(x|y): 0.4998$ - $D(G(z|y)): 0.0289$ - KL Divergence: 4.3214

Epoch 117/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0507 - g_loss: 9.6531 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0258$ - KL Divergence: 4.4813

Epoch 118/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0499 - g_loss: 9.5880 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0257$ - KL Divergence: 4.4493

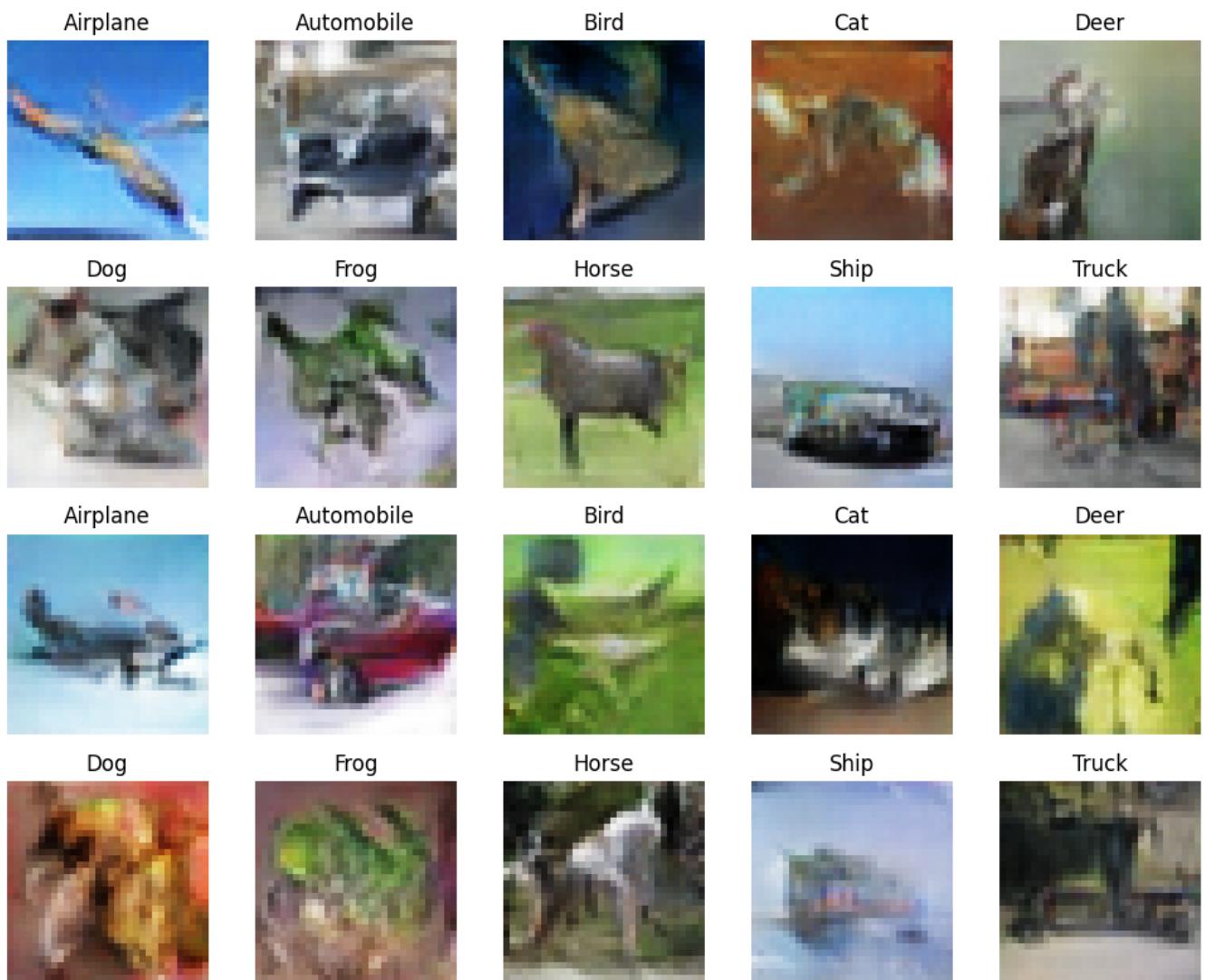
Epoch 119/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0501 - g_loss: 9.5242 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0281$ - KL Divergence: 4.3927

Epoch 120/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0471 - g_loss: 9.4812 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0273$ - KL Divergence: 4.4178

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - ACGAN/generator-epoch-120.h5

Epoch 121/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0497 - g_loss: 9.5106 - D(x|y): 0.5000 - D(G(z|y)): 0.0252 - KL Divergence: 4.4753

Epoch 122/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0535 - g_loss: 9.4896 - D(x|y): 0.5005 - D(G(z|y)): 0.0269 - KL Divergence: 4.4139

Epoch 123/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0487 - g_loss: 9.4986 - D(x|y): 0.5000 - D(G(z|y)): 0.0264 - KL Divergence: 4.5183

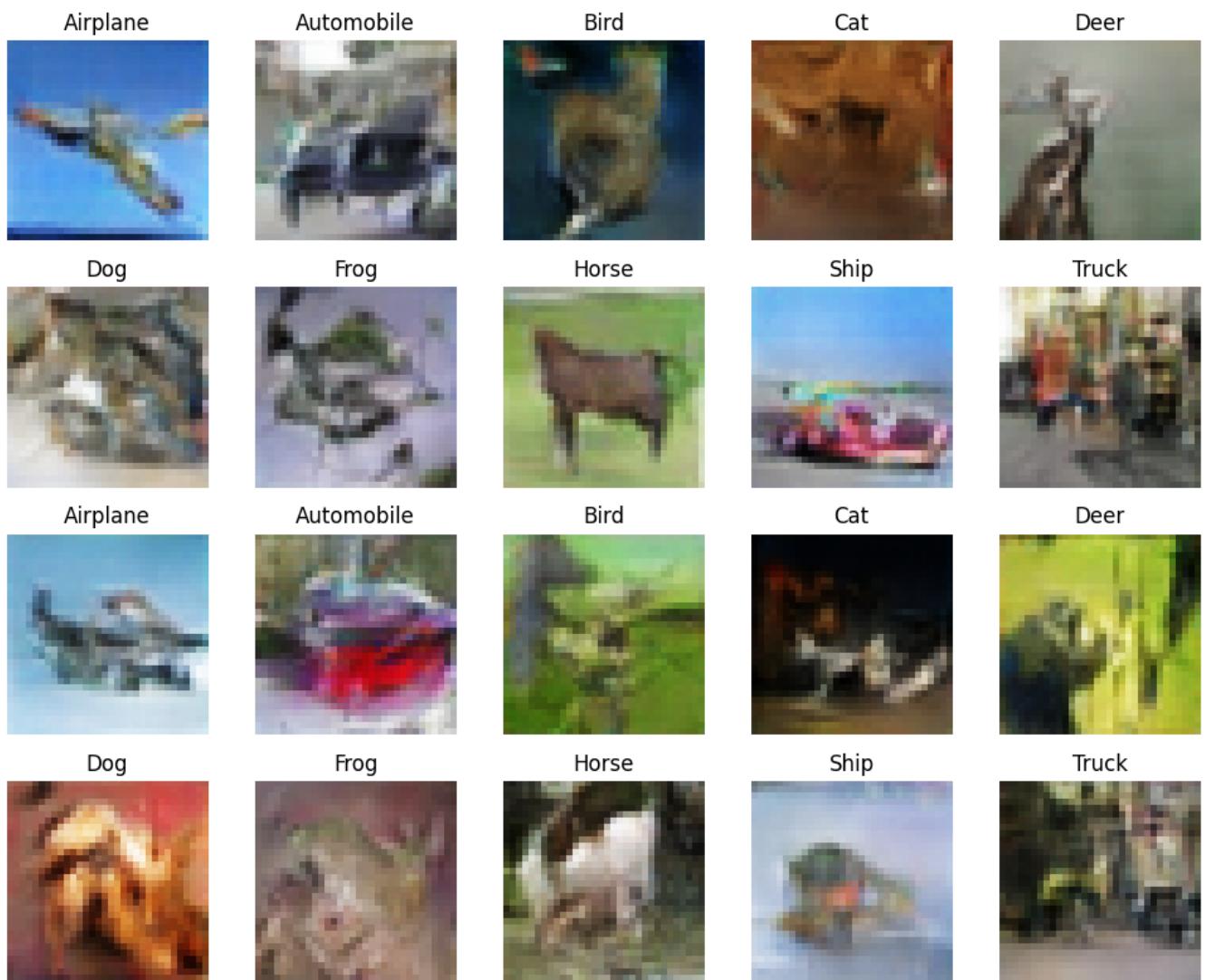
Epoch 124/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0470 - g_loss: 9.5799 - D(x|y): 0.5001 - D(G(z|y)): 0.0269 - KL Divergence: 4.5259

Epoch 125/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0479 - g_loss: 9.5899 - D(x|y): 0.5003 - D(G(z|y)): 0.0250 - KL Divergence: 4.5949

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - ACGAN/generator-epoch-125.h5

Epoch 126/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0484 - g_loss: 9.5200 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0241$ - KL Divergence: 4.6112

Epoch 127/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0461 - g_loss: 9.7063 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0243$ - KL Divergence: 4.5735

Epoch 128/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0483 - g_loss: 9.6515 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0269$ - KL Divergence: 4.6244

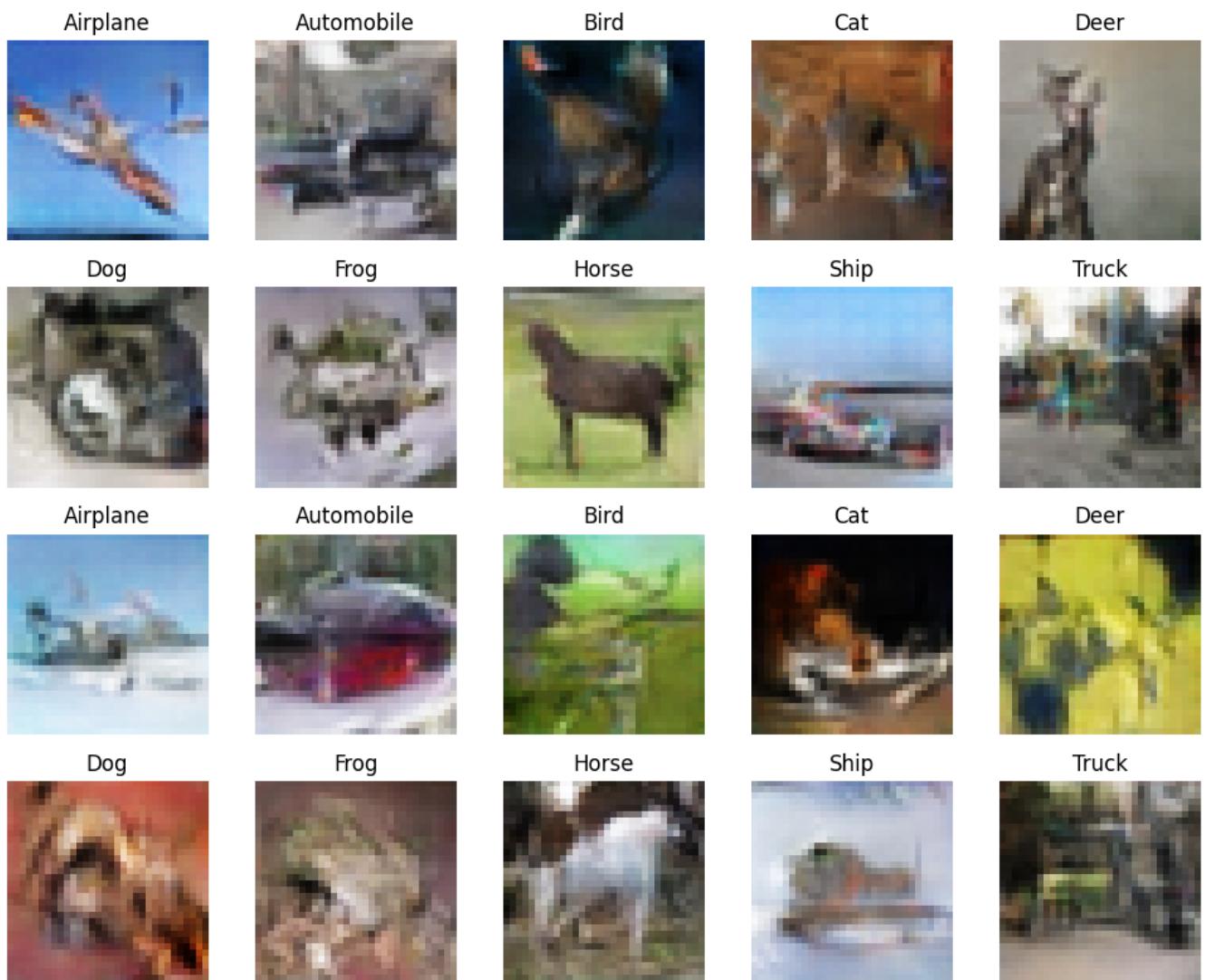
Epoch 129/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0472 - g_loss: 9.6410 -
 $D(x|y): 0.4998$ - $D(G(z|y)): 0.0252$ - KL Divergence: 4.5093

Epoch 130/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0511 - g_loss: 9.7725 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0247$ - KL Divergence: 4.4478

1/1 [=====] - 0s 34ms/step



Generator Checkpoint - ACGAN/generator-epoch-130.h5

Epoch 131/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0483 - g_loss: 9.8853 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0250 - KL Divergence: 4.5073

Epoch 132/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0472 - g_loss: 9.9363 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0233 - KL Divergence: 4.5257

Epoch 133/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0465 - g_loss: 9.9072 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0241 - KL Divergence: 4.4540

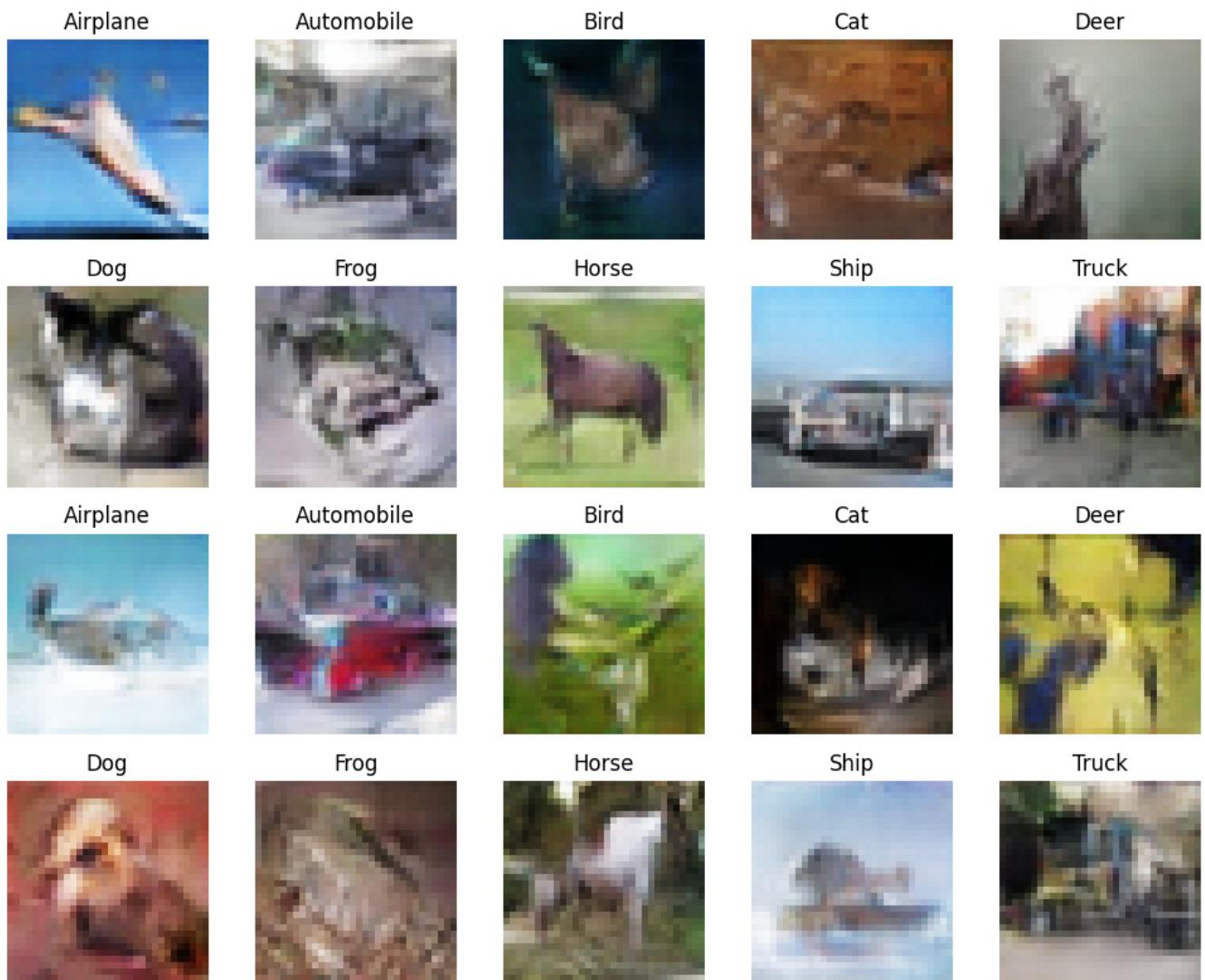
Epoch 134/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0485 - g_loss: 9.9278 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0233 - KL Divergence: 4.5996

Epoch 135/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0433 - g_loss: 10.0673 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0235 - KL Divergence: 4.4603

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - ACGAN/generator-epoch-135.h5

Epoch 136/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0437 - g_loss: 10.0597 -
 $D(x|y): 0.4997$ - $D(G(z|y)): 0.0230$ - KL Divergence: 4.6248

Epoch 137/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0448 - g_loss: 10.1266 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0229$ - KL Divergence: 4.6326

Epoch 138/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0414 - g_loss: 10.1001 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0220$ - KL Divergence: 4.8064

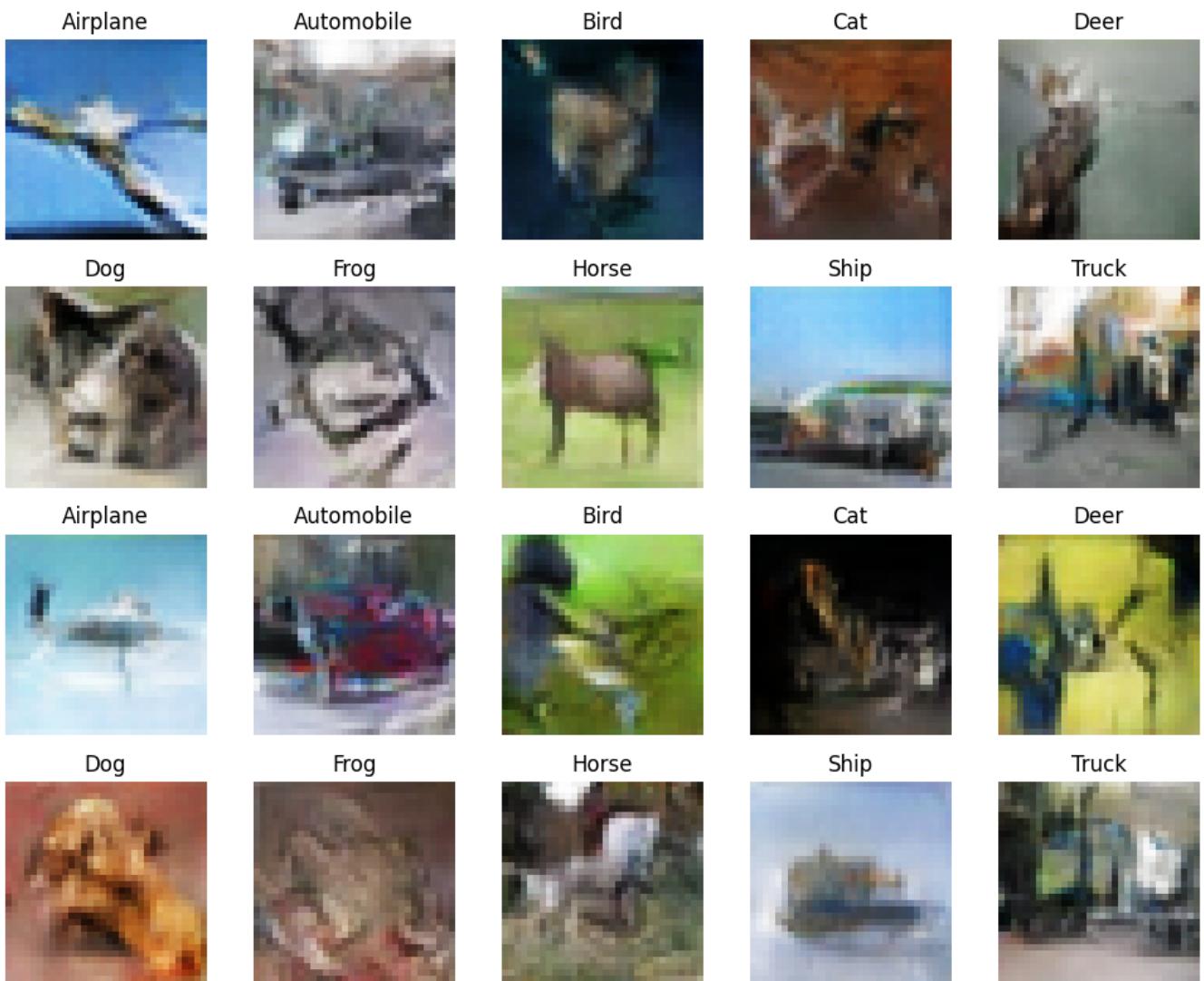
Epoch 139/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0447 - g_loss: 9.9543 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0229$ - KL Divergence: 4.6194

Epoch 140/200

782/782 [=====] - 65s 82ms/step - d_loss: 0.0416 - g_loss: 9.9515 -
 $D(x|y): 0.5005$ - $D(G(z|y)): 0.0237$ - KL Divergence: 4.5918

1/1 [=====] - 0s 22ms/step



Generator Checkpoint - ACGAN/generator-epoch-140.h5

Epoch 141/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0439 - g_loss: 10.1954 - D(x|y): 0.4999 - D(G(z|y)): 0.0224 - KL Divergence: 4.6564

Epoch 142/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0445 - g_loss: 10.1081 - D(x|y): 0.4996 - D(G(z|y)): 0.0244 - KL Divergence: 4.4255

Epoch 143/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0427 - g_loss: 10.2899 - D(x|y): 0.5003 - D(G(z|y)): 0.0226 - KL Divergence: 4.5401

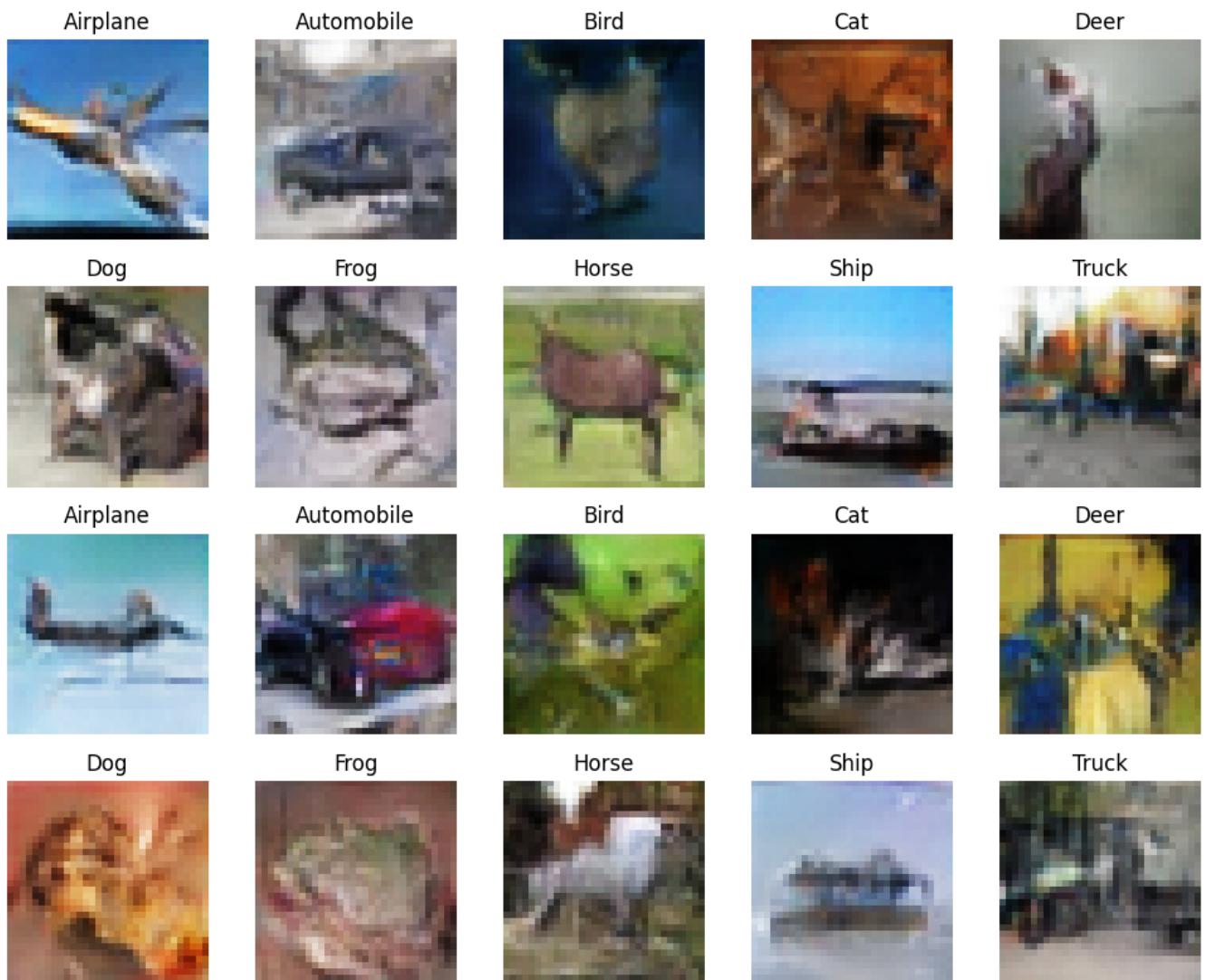
Epoch 144/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0415 - g_loss: 10.2155 - D(x|y): 0.4999 - D(G(z|y)): 0.0209 - KL Divergence: 4.6224

Epoch 145/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0396 - g_loss: 10.2334 - D(x|y): 0.5002 - D(G(z|y)): 0.0228 - KL Divergence: 4.6463

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - ACGAN/generator-epoch-145.h5

Epoch 146/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0412 - g_loss: 10.2631 - D(x|y): 0.5004 - D(G(z|y)): 0.0221 - KL Divergence: 4.6867

Epoch 147/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0410 - g_loss: 10.3057 - D(x|y): 0.5001 - D(G(z|y)): 0.0204 - KL Divergence: 4.6379

Epoch 148/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0419 - g_loss: 10.2060 - D(x|y): 0.5000 - D(G(z|y)): 0.0221 - KL Divergence: 4.5885

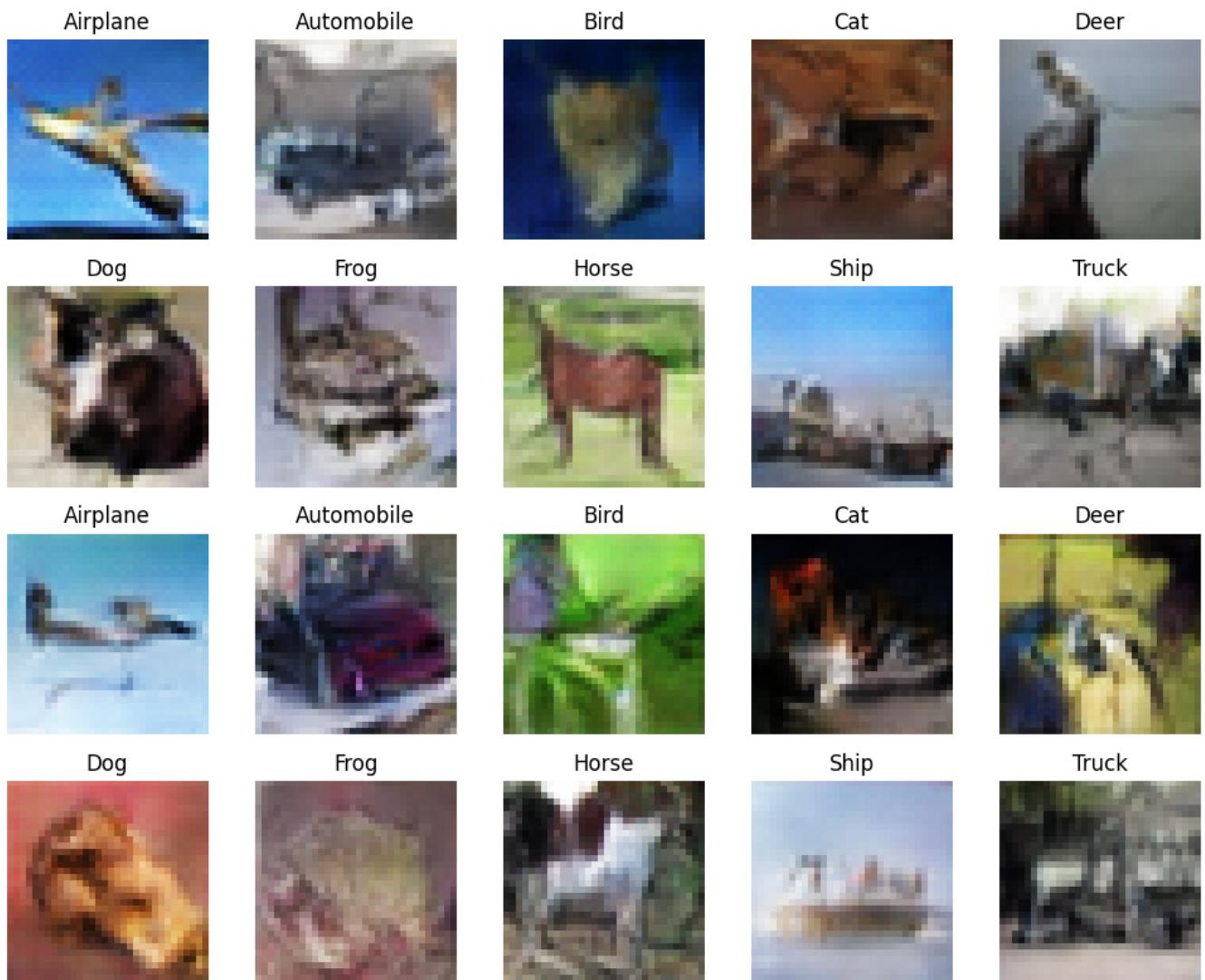
Epoch 149/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0399 - g_loss: 10.3822 - D(x|y): 0.5003 - D(G(z|y)): 0.0225 - KL Divergence: 4.7194

Epoch 150/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0380 - g_loss: 10.3306 - D(x|y): 0.5002 - D(G(z|y)): 0.0211 - KL Divergence: 4.5707

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-150.h5

Epoch 151/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0382 - g_loss: 10.4163 - D(x|y): 0.5002 - D(G(z|y)): 0.0217 - KL Divergence: 4.4491

Epoch 152/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0388 - g_loss: 10.5753 - D(x|y): 0.5002 - D(G(z|y)): 0.0202 - KL Divergence: 4.5556

Epoch 153/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0407 - g_loss: 10.4754 - D(x|y): 0.5001 - D(G(z|y)): 0.0213 - KL Divergence: 4.4323

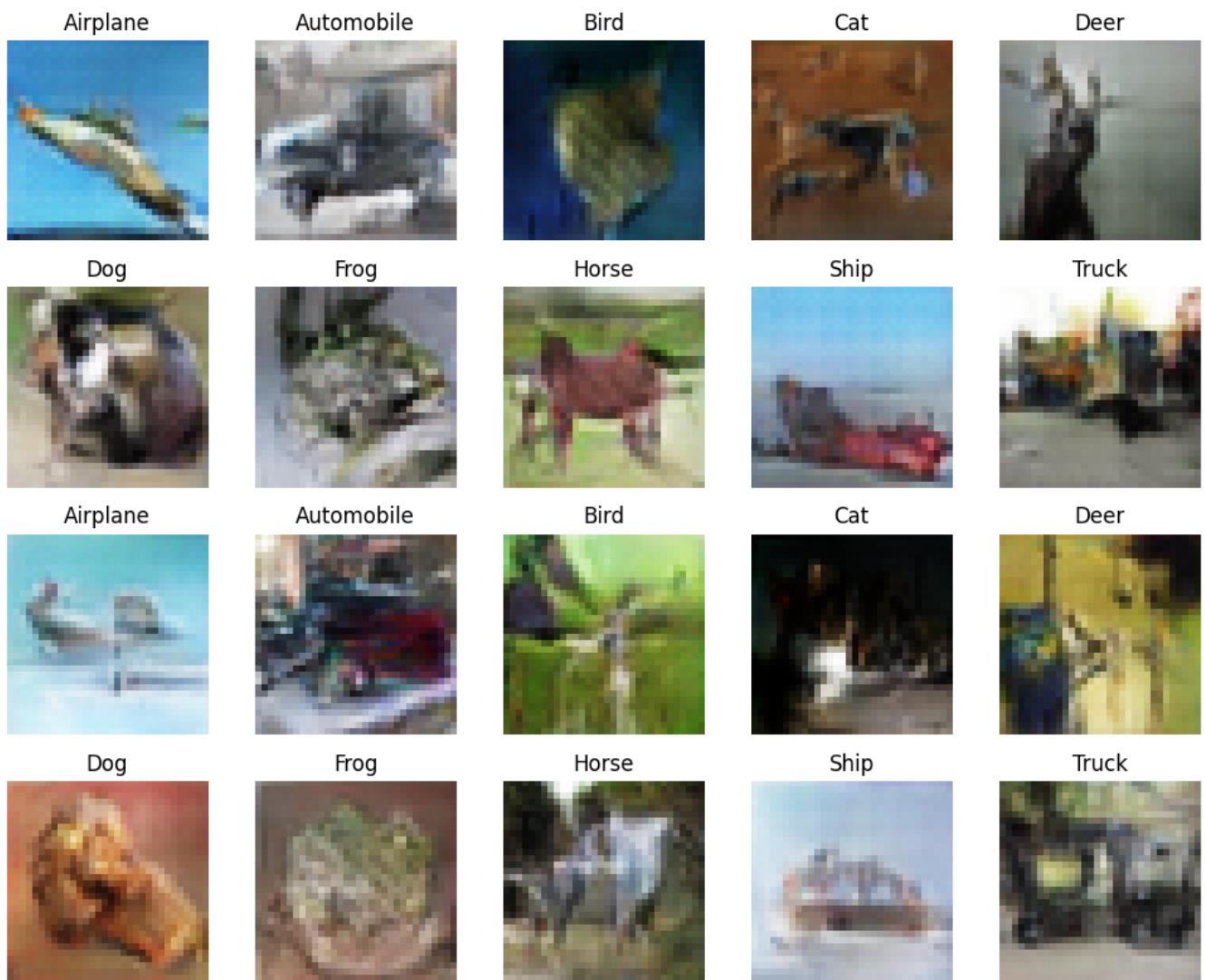
Epoch 154/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0414 - g_loss: 10.4811 - D(x|y): 0.5002 - D(G(z|y)): 0.0191 - KL Divergence: 4.4706

Epoch 155/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0360 - g_loss: 10.4229 - D(x|y): 0.5002 - D(G(z|y)): 0.0208 - KL Divergence: 4.5416

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - ACGAN/generator-epoch-155.h5

Epoch 156/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0391 - g_loss: 10.5886 - D(x|y): 0.5002 - D(G(z|y)): 0.0212 - KL Divergence: 4.5294

Epoch 157/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0356 - g_loss: 10.5692 - D(x|y): 0.5001 - D(G(z|y)): 0.0188 - KL Divergence: 4.6173

Epoch 158/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0344 - g_loss: 10.5680 - D(x|y): 0.5000 - D(G(z|y)): 0.0204 - KL Divergence: 4.6693

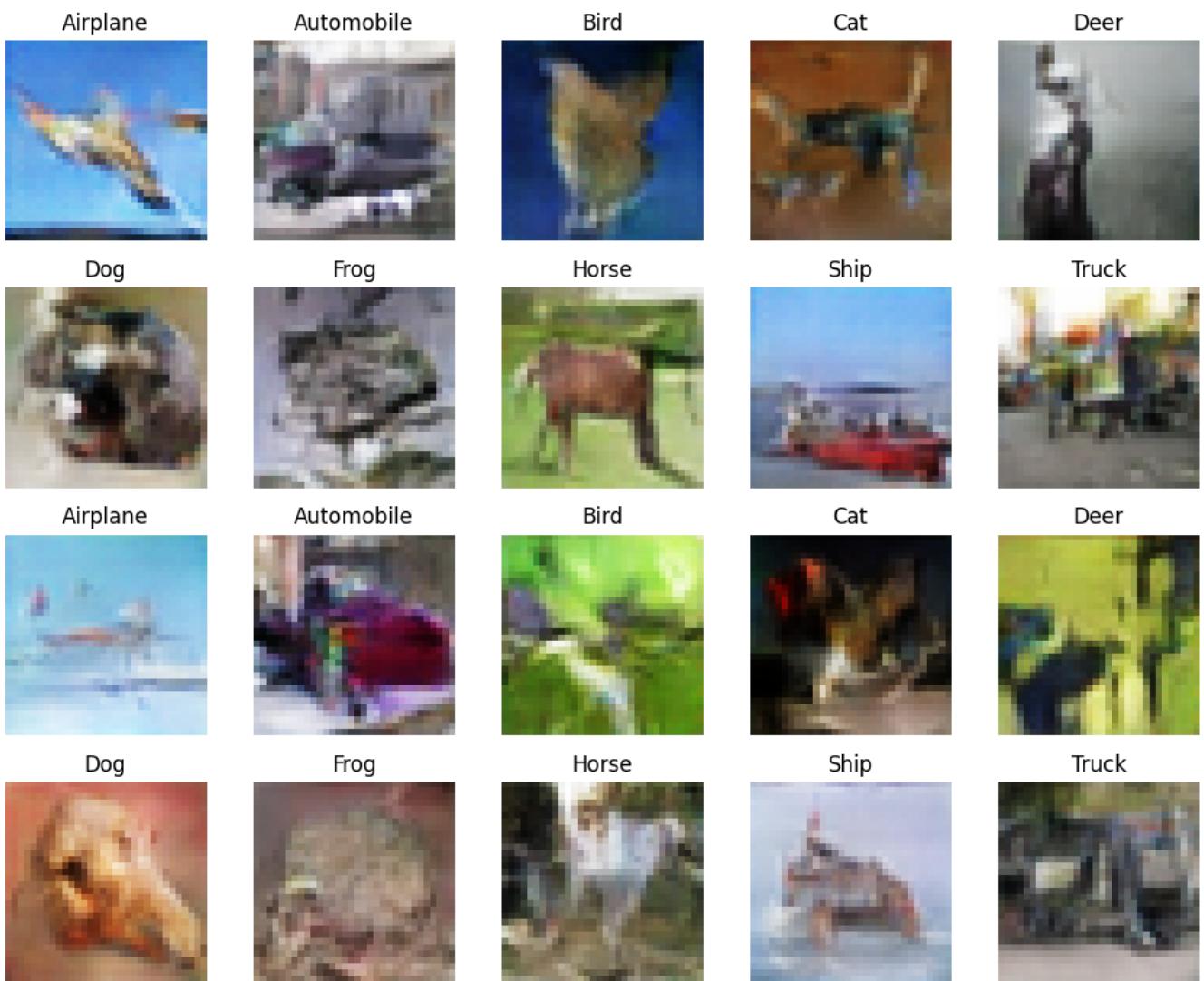
Epoch 159/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0381 - g_loss: 10.3157 - D(x|y): 0.5002 - D(G(z|y)): 0.0209 - KL Divergence: 4.5108

Epoch 160/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0389 - g_loss: 10.5065 - D(x|y): 0.5002 - D(G(z|y)): 0.0211 - KL Divergence: 4.4974

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - ACGAN/generator-epoch-160.h5

Epoch 161/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0389 - g_loss: 10.3096 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0197 - KL Divergence: 4.5462

Epoch 162/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0380 - g_loss: 10.5626 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0218 - KL Divergence: 4.4762

Epoch 163/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0390 - g_loss: 10.5609 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0208 - KL Divergence: 4.5961

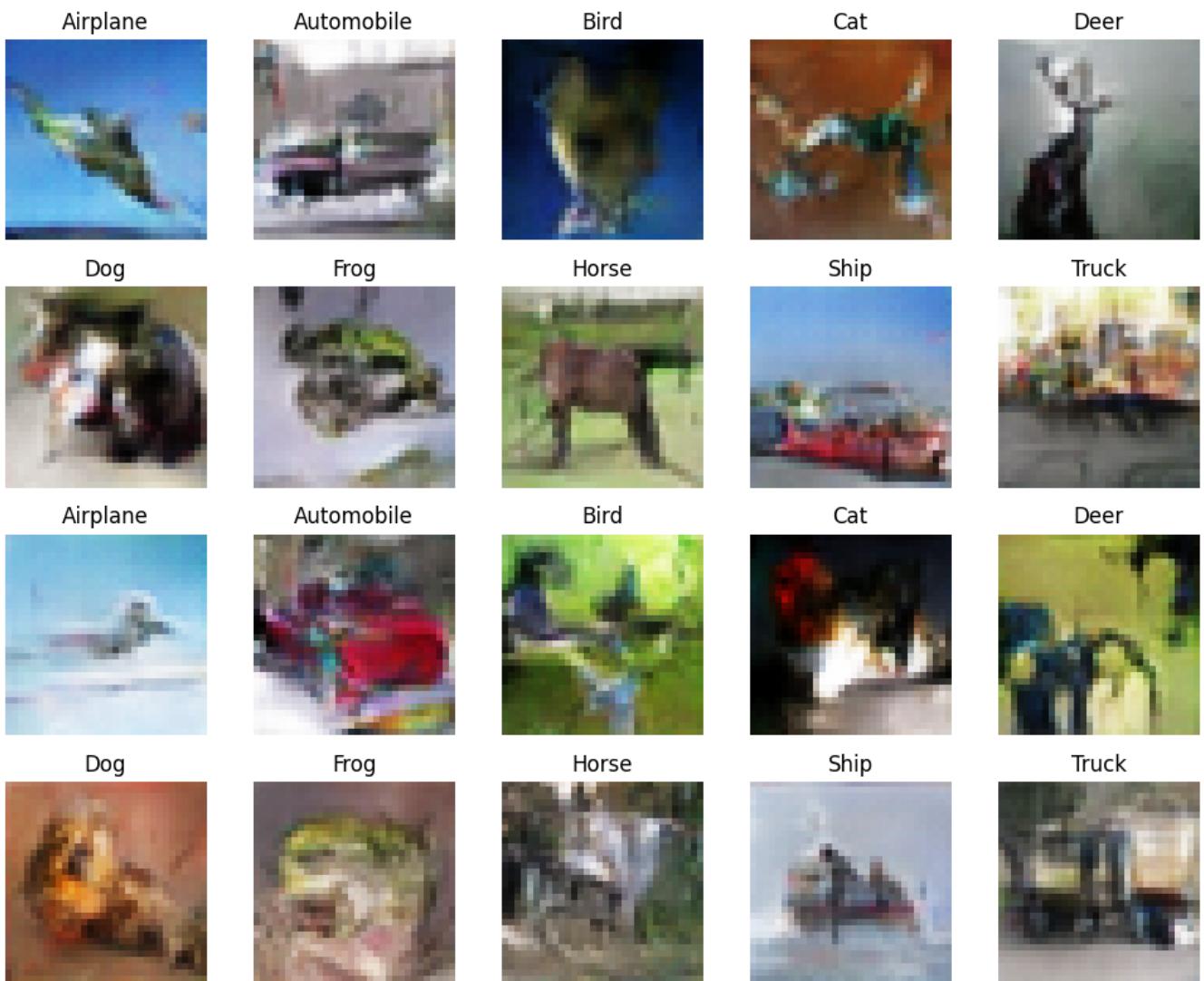
Epoch 164/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0377 - g_loss: 10.6561 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0205 - KL Divergence: 4.4667

Epoch 165/200

782/782 [=====] - 65s 82ms/step - d_loss: 0.0391 - g_loss: 10.3624 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0193 - KL Divergence: 4.5474

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-165.h5

Epoch 166/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0376 - g_loss: 10.8100 -
 $D(x|y): 0.4998$ - $D(G(z|y)): 0.0191$ - KL Divergence: 4.3804

Epoch 167/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0341 - g_loss: 10.7767 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0193$ - KL Divergence: 4.4211

Epoch 168/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0344 - g_loss: 10.5924 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0191$ - KL Divergence: 4.5924

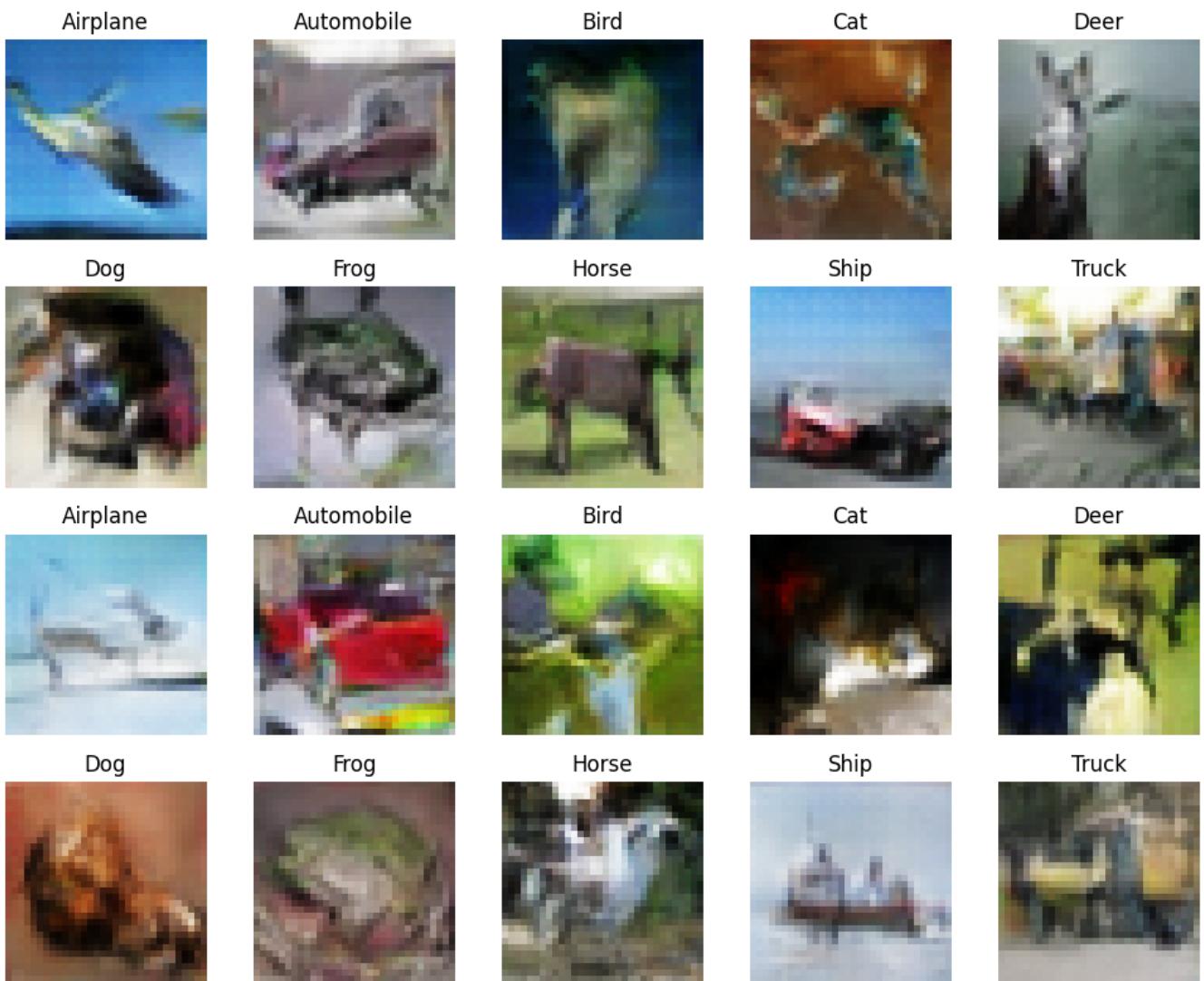
Epoch 169/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0372 - g_loss: 10.6332 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0194$ - KL Divergence: 4.5628

Epoch 170/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0344 - g_loss: 10.6284 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0184$ - KL Divergence: 4.5610

1/1 [=====] - 0s 29ms/step



Generator Checkpoint - ACGAN/generator-epoch-170.h5

Epoch 171/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0389 - g_loss: 10.6685 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0200 - KL Divergence: 4.5662

Epoch 172/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0385 - g_loss: 10.4595 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0204 - KL Divergence: 4.6869

Epoch 173/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0330 - g_loss: 10.7177 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0178 - KL Divergence: 4.5463

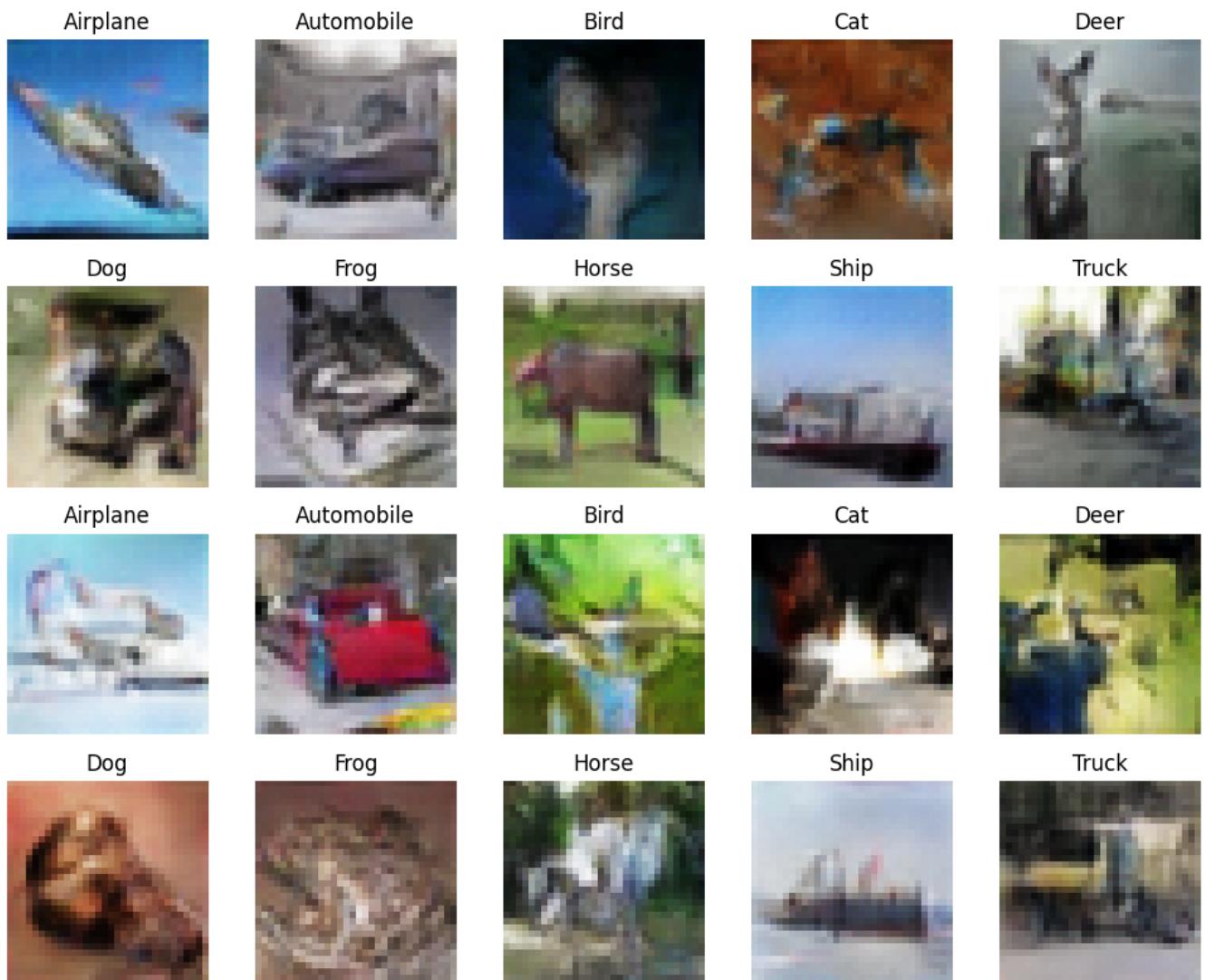
Epoch 174/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0348 - g_loss: 10.8425 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0186 - KL Divergence: 4.5929

Epoch 175/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0354 - g_loss: 10.8588 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0183 - KL Divergence: 4.4169

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - ACGAN/generator-epoch-175.h5

Epoch 176/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0356 - g_loss: 10.7409 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0180$ - KL Divergence: 4.6203

Epoch 177/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0345 - g_loss: 10.6794 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.0189$ - KL Divergence: 4.6679

Epoch 178/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0379 - g_loss: 10.7066 -
 $D(x|y): 0.4997$ - $D(G(z|y)): 0.0199$ - KL Divergence: 4.6324

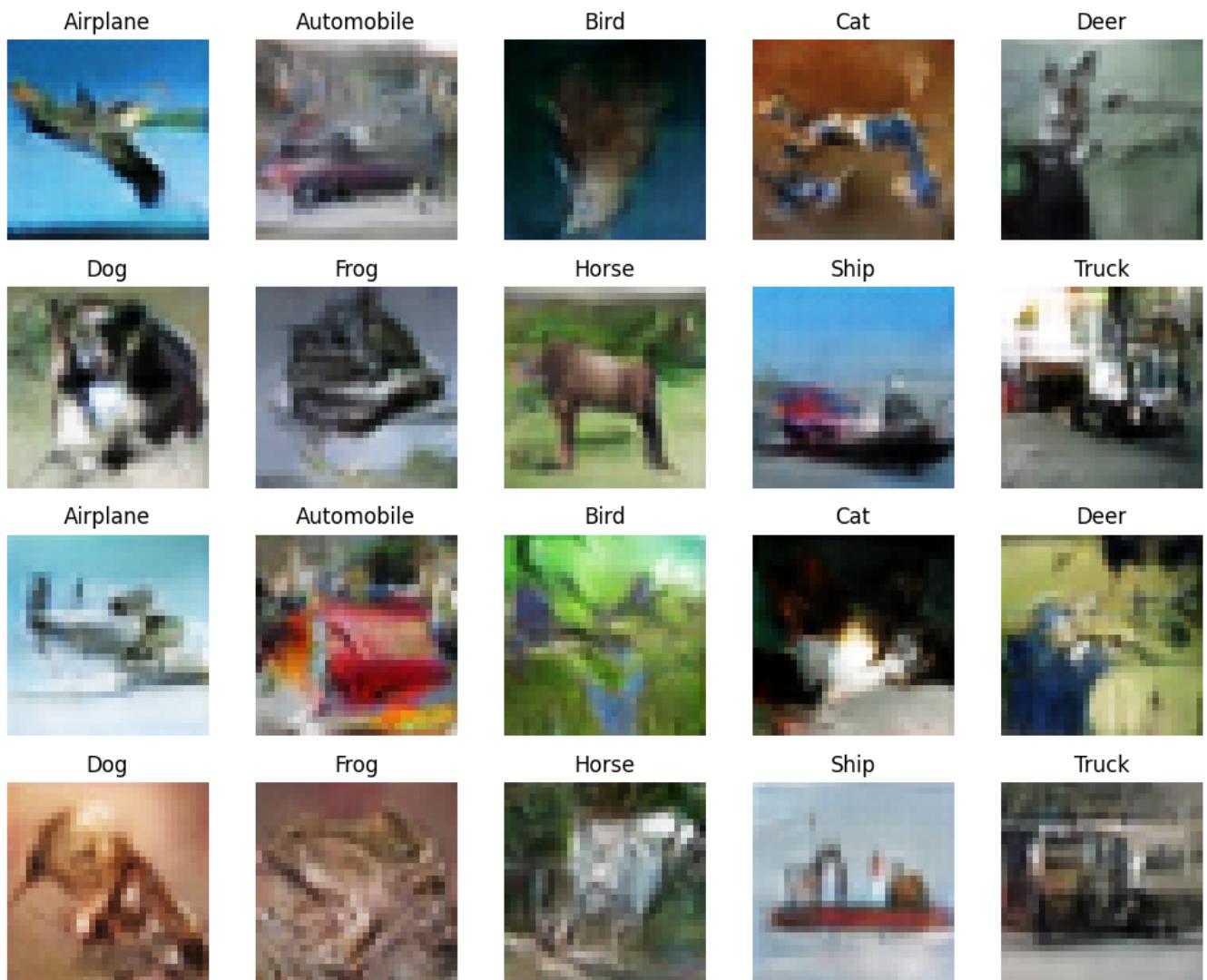
Epoch 179/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0337 - g_loss: 11.0204 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0171$ - KL Divergence: 4.4794

Epoch 180/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0362 - g_loss: 10.9560 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.0187$ - KL Divergence: 4.5774

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - ACGAN/generator-epoch-180.h5

Epoch 181/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0348 - g_loss: 11.0296 - D(x|y): 0.5002 - D(G(z|y)): 0.0182 - KL Divergence: 4.6194

Epoch 182/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0311 - g_loss: 11.1445 - D(x|y): 0.5002 - D(G(z|y)): 0.0176 - KL Divergence: 4.5359

Epoch 183/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.0313 - g_loss: 10.9244 - D(x|y): 0.5002 - D(G(z|y)): 0.0177 - KL Divergence: 4.4815

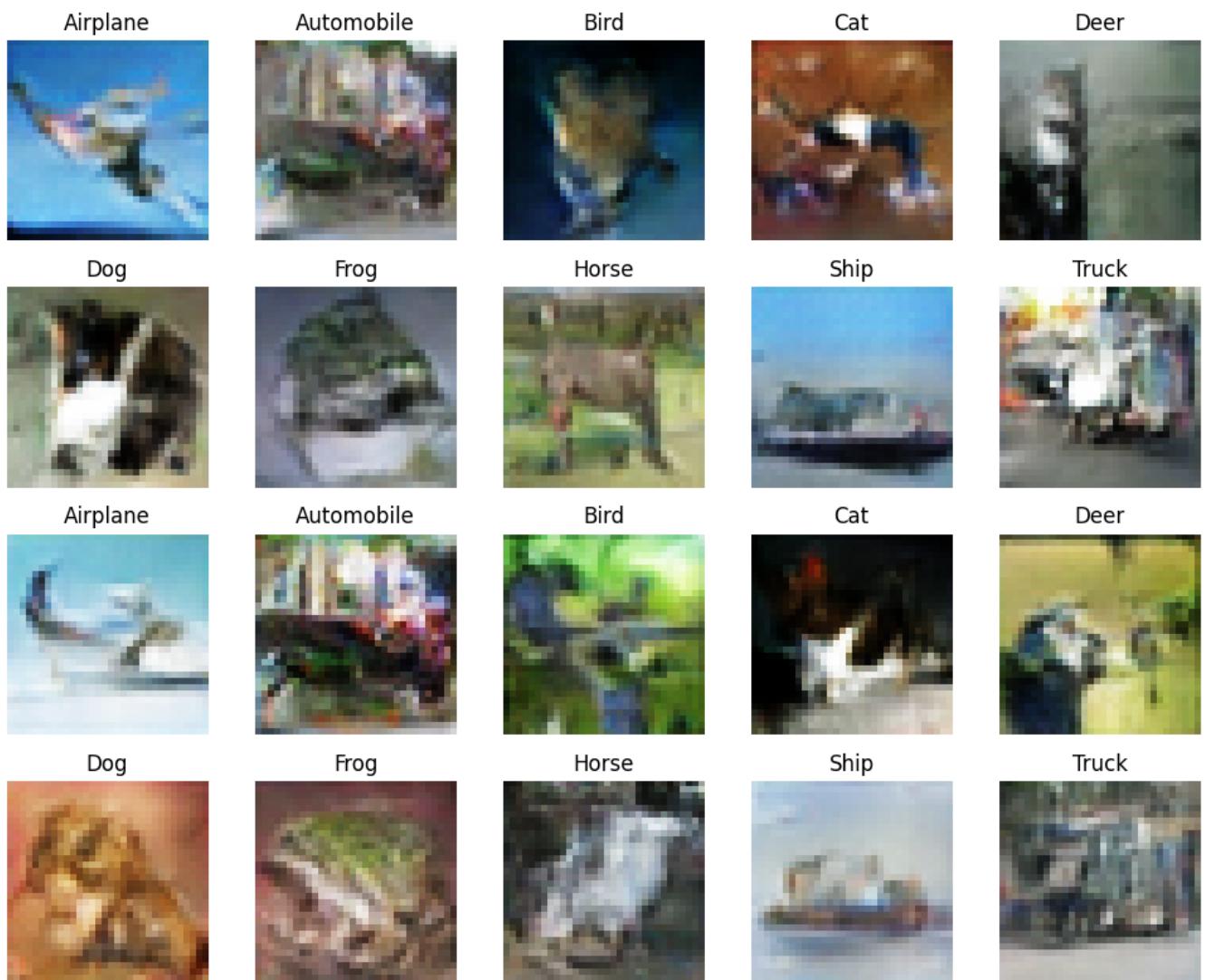
Epoch 184/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0388 - g_loss: 10.6597 - D(x|y): 0.4999 - D(G(z|y)): 0.0186 - KL Divergence: 4.6581

Epoch 185/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0353 - g_loss: 10.8569 - D(x|y): 0.5000 - D(G(z|y)): 0.0179 - KL Divergence: 4.5563

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - ACGAN/generator-epoch-185.h5

Epoch 186/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0356 - g_loss: 10.5855 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0170$ - KL Divergence: 4.6060

Epoch 187/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0414 - g_loss: 10.3453 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.0181$ - KL Divergence: 4.4798

Epoch 188/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0391 - g_loss: 10.6659 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0177$ - KL Divergence: 4.7812

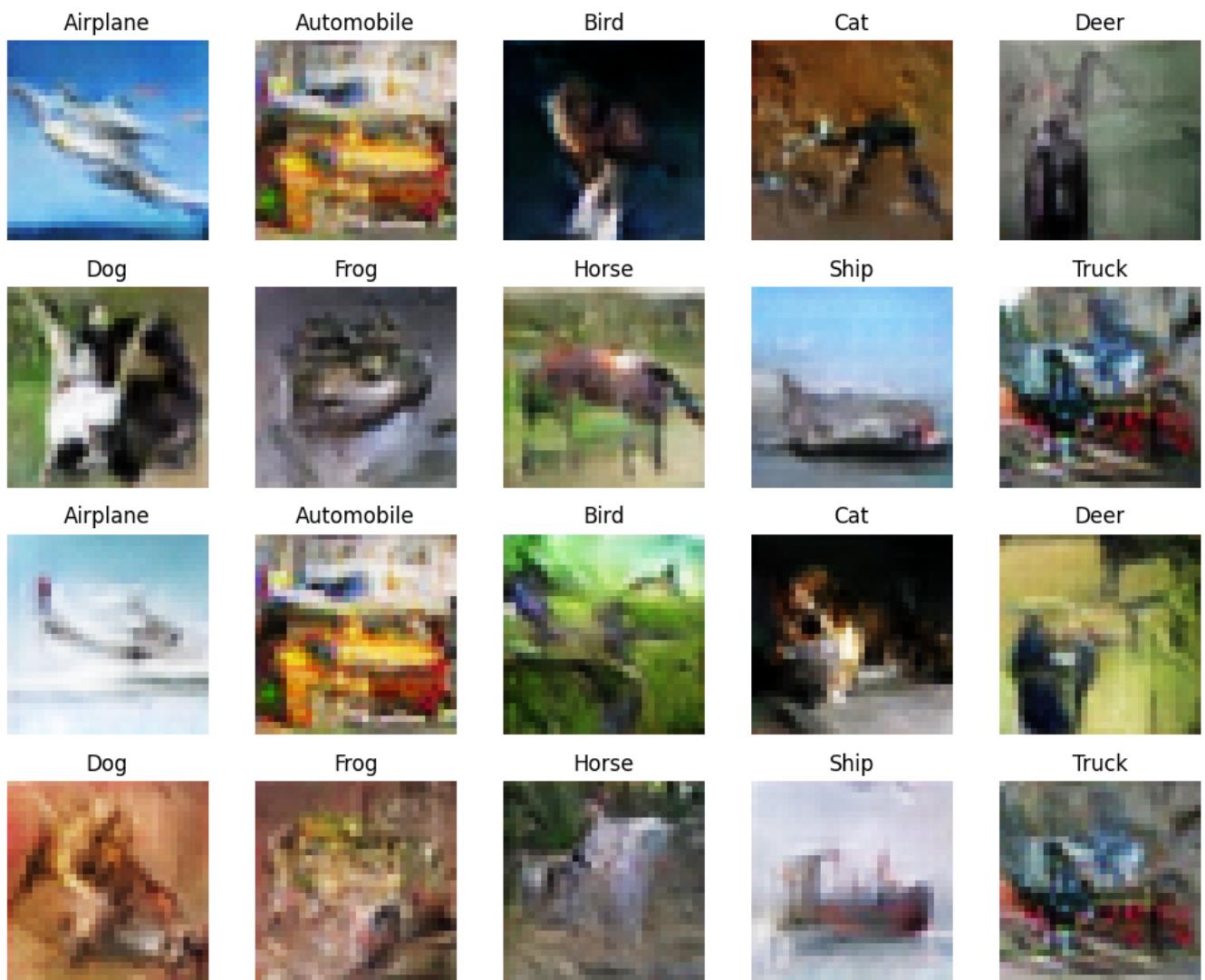
Epoch 189/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0378 - g_loss: 10.2462 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.0174$ - KL Divergence: 4.6804

Epoch 190/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0401 - g_loss: 9.8650 -
 $D(x|y): 0.5007$ - $D(G(z|y)): 0.0180$ - KL Divergence: 4.9386

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - ACGAN/generator-epoch-190.h5

Epoch 191/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0374 - g_loss: 9.9178 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.0188 - KL Divergence: 4.7206

Epoch 192/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0426 - g_loss: 10.2274 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0184 - KL Divergence: 4.7145

Epoch 193/200

782/782 [=====] - 68s 88ms/step - d_loss: 0.0405 - g_loss: 9.9893 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0176 - KL Divergence: 4.2209

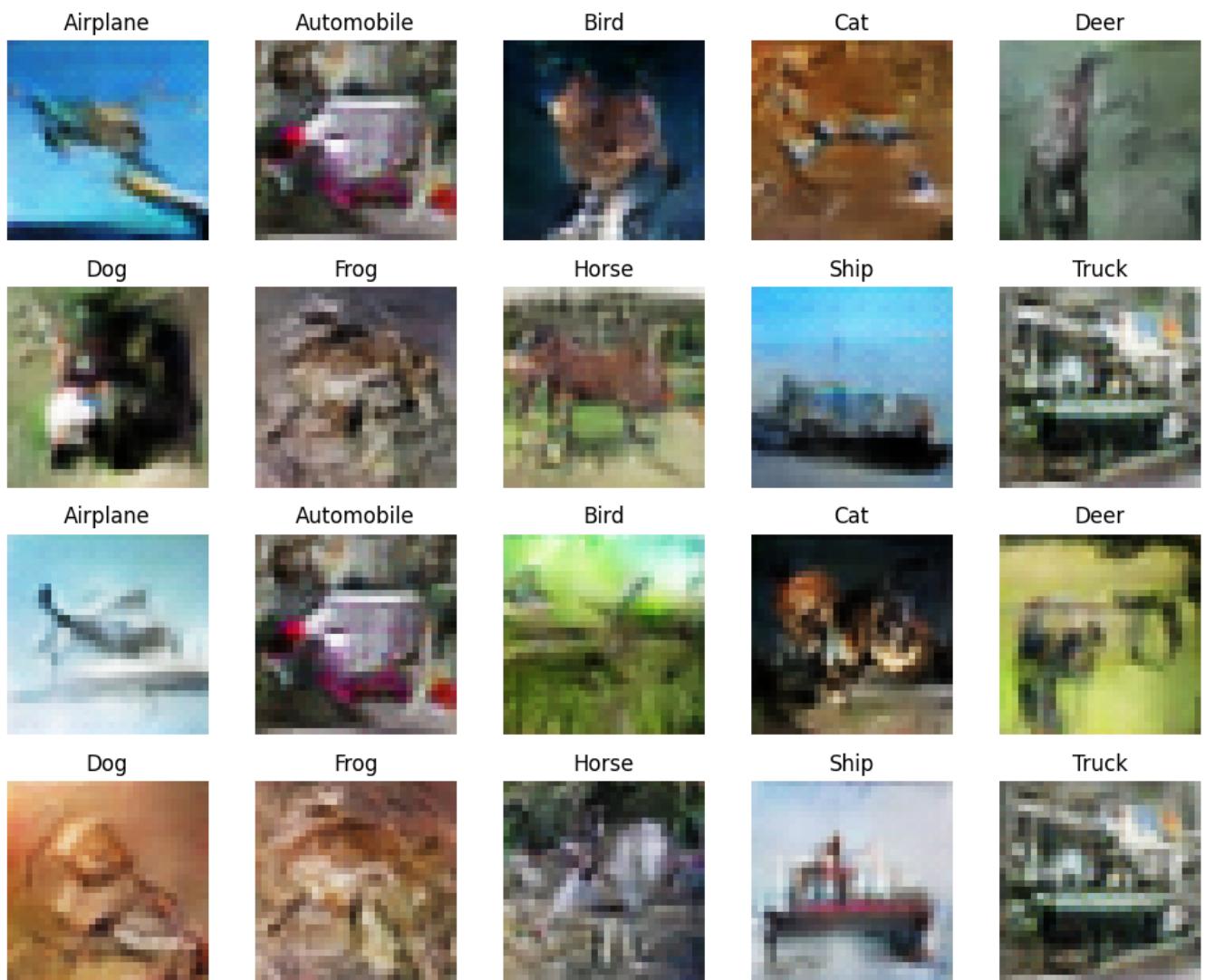
Epoch 194/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0451 - g_loss: 10.1866 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0184 - KL Divergence: 4.1549

Epoch 195/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0426 - g_loss: 9.9711 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0198 - KL Divergence: 4.4706

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - ACGAN/generator-epoch-195.h5

Epoch 196/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0433 - g_loss: 9.8577 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0206$ - KL Divergence: 4.8519

Epoch 197/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.0470 - g_loss: 9.9979 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0196$ - KL Divergence: 4.9123

Epoch 198/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0490 - g_loss: 9.8164 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0194$ - KL Divergence: 4.2961

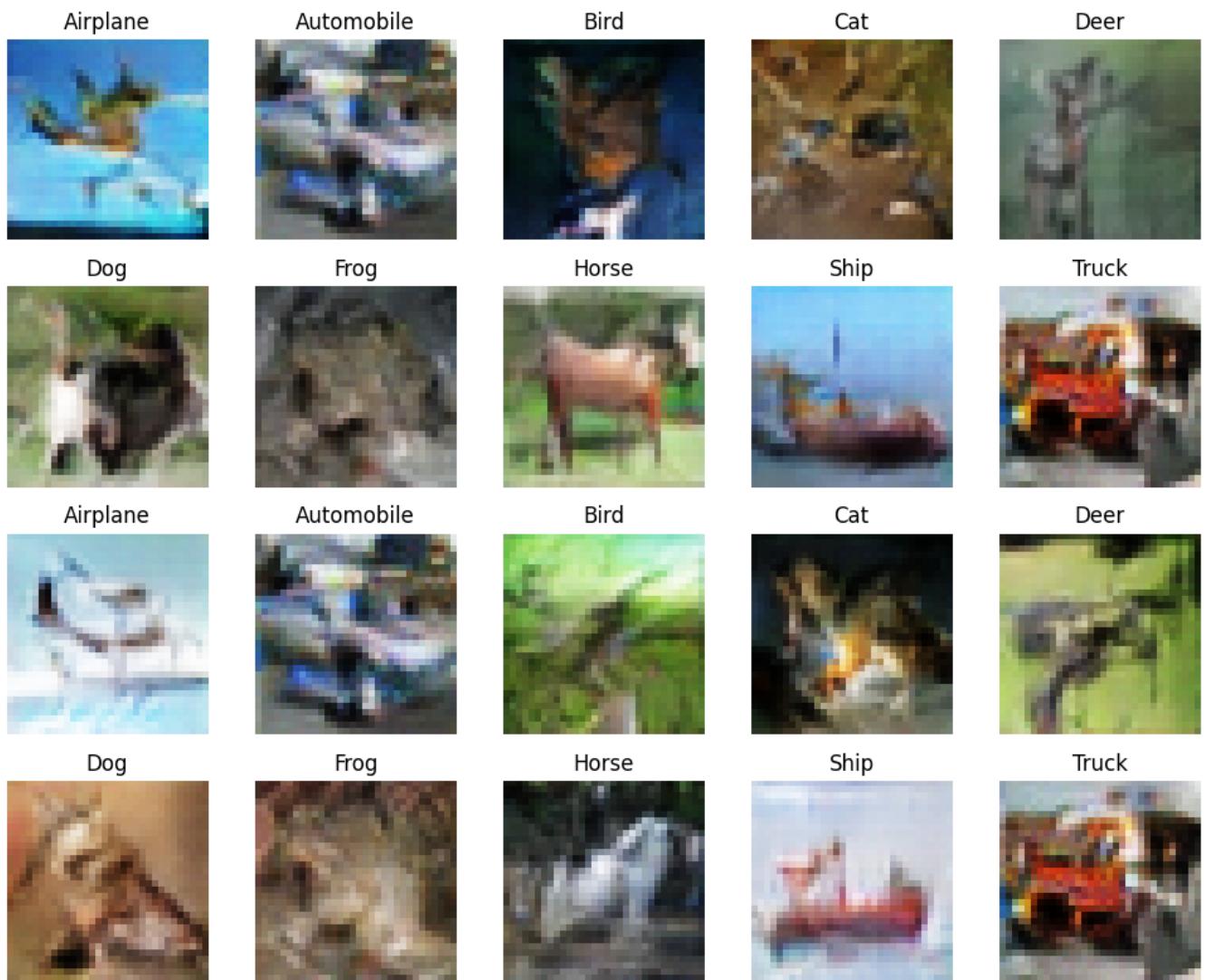
Epoch 199/200

782/782 [=====] - 69s 88ms/step - d_loss: 0.0519 - g_loss: 9.6917 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.0216$ - KL Divergence: 4.1689

Epoch 200/200

782/782 [=====] - 68s 87ms/step - d_loss: 0.0477 - g_loss: 9.4337 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0199$ - KL Divergence: 4.4272

1/1 [=====] - 0s 32ms/step



Generator Checkpoint - ACGAN/generator-epoch-Full Train.h5

Observations

Similarly to the cGAN, the ACGAN has a issue with same images generating again using the same label and noise. However, this problem only occurred during the later part of the training around 185 - 190. We also notice that the images around 126, there are noise inputs are showing slightly

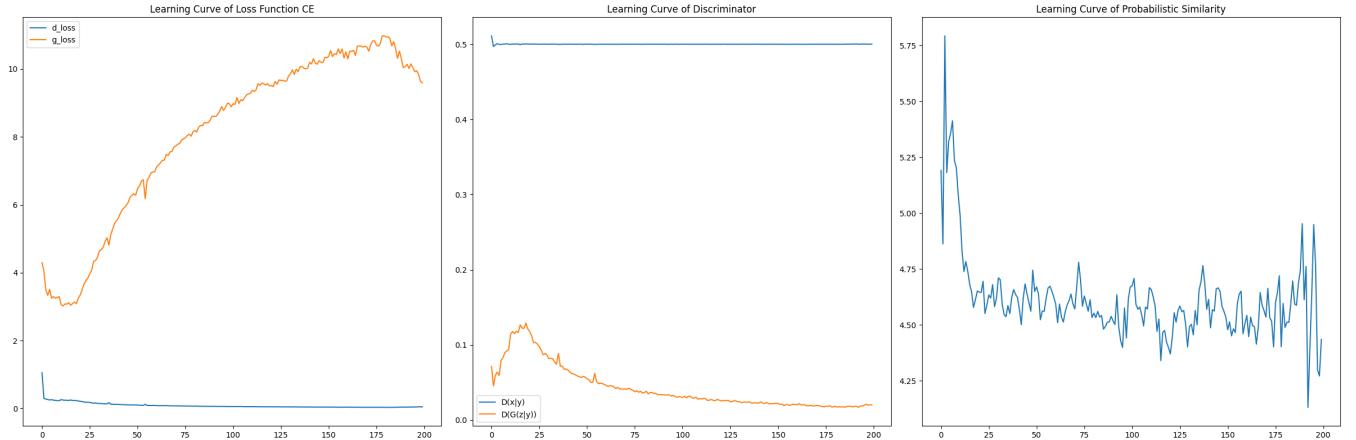
ACGAN Evaluation

As mentioned previously we will be using a quantitative and manual evaluation of the ACGAN.

We will start off with a quantitative analysis so that we can find the best model to generate the images for manual evaluation.

```
In [ ]: # store history object into dataframe
aux_cond_hist_df = pd.DataFrame(aux_cond_hist.history)

# using pandas dataframe to plot out Learning curve
fig, (ax1, ax2, ax3) = plt.subplots(
    1, 3, figsize=(24, 8), tight_layout=True)
aux_cond_hist_df.loc[:, ["d_loss", 'g_loss']].plot(
    ax=ax1, title=r'Learning Curve of Loss Function CE')
aux_cond_hist_df.loc[:, ['D(x|y)', 'D(G(z|y))']].plot(
    ax=ax2, title=r'Learning Curve of Discriminator')
aux_cond_hist_df.loc[:, 'KL Divergence'].plot(
    ax=ax3, title=r'Learning Curve of Probabilistic Similarity')
plt.show()
```



Observations

We note that as the model has collapsed early around 185 epochs. The lowest KL divergence is 115 epochs.

```
In [ ]: # Loading Weights for best Generator
saved_weights = 'ACGAN\generator-epoch-115.h5'
aux_cond_gan.generator.load_weights(saved_weights)
aux_cond_gan.generator.summary()
```

Model: "generator_cGAN"

Layer (type)	Output Shape	Param #	Connected to
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_16 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_14 (Dense)	(None, 2048)	284672	['concatenate_16[0][0]']
batch_normalization_150 (Batch Normalization)	(None, 2048)	8192	['dense_14[0][0]']

Layer (type)	Output Shape	Param #	Connected to
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_16 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_14 (Dense)	(None, 2048)	284672	['concatenate_16[0][0]']
batch_normalization_150 (Batch Normalization)	(None, 2048)	8192	['dense_14[0][0]']
leaky_re_lu_40 (LeakyReLU)	(None, 2048)	0	['batch_normalization_150[0][0]']
reshape_5 (Reshape)	(None, 2, 2, 512)	0	['leaky_re_lu_40[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2754752	['reshape_5[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']

=====

Total params: 3,050,691
Trainable params: 3,045,699
Non-trainable params: 4,992

In []:

```
n = 10000

# generating labels
labels = np.random.randint(low=0, high=10, size=n)
one_hot_labels = to_categorical(labels)

# Generating 1000 Synthetic Images
random_noise = tf.random.normal(shape=(n, NOISE))

synthetic_images = aux_cond_gan.generator.predict(
    [random_noise, one_hot_labels])
```

```

print("Latent Vector Dim: {} \t Generated Images Dim: {}".format(
    random_noise.shape, synthetic_images.shape))

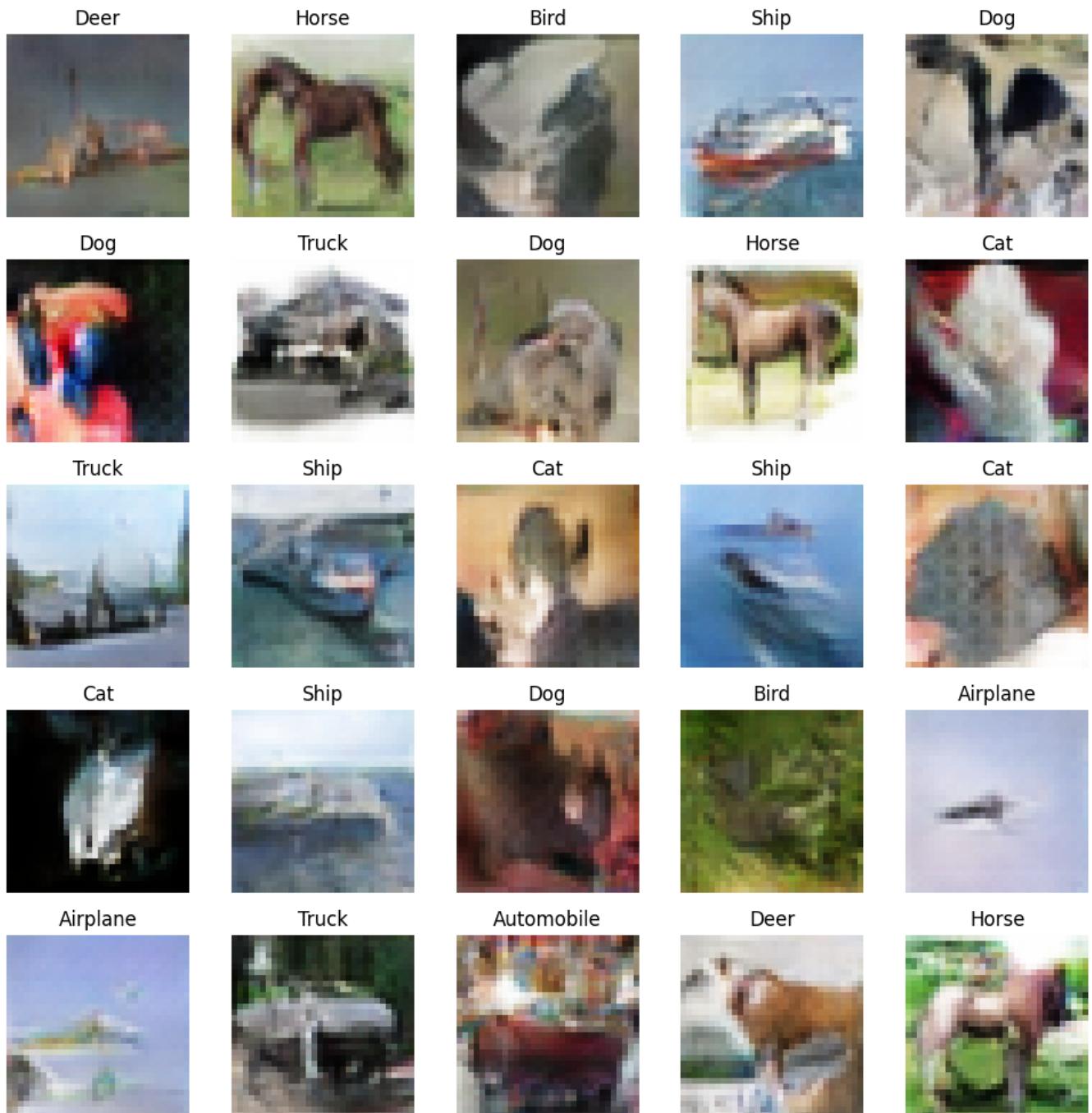
# Scaling back to [0, 1]
synthetic_images -= -1
synthetic_images /= (1 - (-1))

# Display 25 randomly sampled images
fig = plt.figure(figsize=(10, 10), tight_layout=True)
for i in range(25):
    rand_idx = np.random.randint(0, len(synthetic_images))
    ax = fig.add_subplot(5, 5, i+1)
    ax.imshow(synthetic_images[rand_idx])
    ax.set_title(class_labels[labels[rand_idx]])
    ax.axis('off')
plt.show()

```

313/313 [=====] - 4s 11ms/step

Latent Vector Dim: (10000, 128) Generated Images Dim: (10000, 32, 32, 3)



Observations

We can see the generator being able to more clearly draw out the vehicles as well as same of the animals like deer and horse. However, some of the animals like cat and dog etc are still very distorted.

Some changes should be made to the architecture like introducing spectral normalisation or regularisation to allow the model to generalise and learn features faster.

```
In [ ]: aux_cond_gan_fid_class = GAN_FID(batch_size=512, noise=128,
                                         sample_size=10000, buffer_size=1024, labels=True)
aux_cond_gan_fid_class.fit(
    generator=aux_cond_gan.generator, train_data=x_train)
fid_score = aux_cond_gan_fid_class.evaluate()

Computing Real Image Embeddings
 0% | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 6s 122ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 126ms/step
16/16 [=====] - 2s 125ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 126ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 129ms/step
Computing Generated Image Embeddings
 0% | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 134ms/step
16/16 [=====] - 2s 133ms/step
Computed Embeddings      Real Images Embedding Shape: (10240, 2048)      Generated Images Embe
dding Shape: (10240, 2048)
The computed FID score is: 88.7613090961658
```

Observations

We note that the FID score is now lower and drop from 94.536161025068 to 88.7613090961658. This suggest that more needs to be done to allow the FID score to be decreased.

Model Selection

Based on the different model architecture that we have tested, we can see the more complex ACGAN performed better as it is able to have clear images of more classes than the cGAN and DCGAN. It is also able to have a lower FID score compared to both cGAN and DCGAN. However, as FID score is not a good indication for the model generation we will be also improving the cGAN model.

Model Improvements

ACGAN

New ACGAN Discriminator

As mentioned in the DCGAN discriminator creation, we will be using Gaussian Weights layers for the new ACGAN model.

Gaussian weights will reduce the instability of training [mean = 0, std = 0.02].

```
In [ ]: # function to create discriminator model
def create_improve_ACGAN_discriminator(image_size):
    # input image
    img_input = Input(shape=(image_size), name='Image_Input')
    weights_init = RandomNormal(mean=0, stddev=0.02)

    # conditions y
    conditions = Input(shape=(10,), name='Conditions_y')

    base_discriminator = Sequential([
        Conv2D(64, kernel_size=4, strides=2, padding='same',
               kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        Conv2D(128, kernel_size=4, strides=2, padding='same',
               kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        Conv2D(256, kernel_size=4, strides=2, padding='same',
               kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        Conv2D(512, kernel_size=4, strides=2, padding='same',
               kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
    ], name='Base_Discriminator')
    discriminator = base_discriminator(img_input)

    discriminator = GlobalAveragePooling2D()(discriminator)

    # Concatenate - combine with conditions y
    merged_layer = Concatenate()([discriminator, conditions])
    discriminator = Dense(512, activation='relu')(merged_layer)

    # Output
    Disc_Output_Layer = Dense(1, activation='sigmoid',
                               name='Disc_Output_Layer')(discriminator)
    Aux_Output_Layer = Dense(10, activation='softmax',
                            name='Aux_Output_Layer')(discriminator)

    discriminator = Model(inputs=[img_input, conditions],
                          outputs=[Disc_Output_Layer, Aux_Output_Layer], name='discriminator_')
```

```
    return discriminator
```

```
create_improve_ACGAN_discriminator(image_size=IMG_SIZE).summary()
```

```
c:\Users\Soh Hong Yu\anaconda3\envs\gpu_env\lib\site-packages\keras\initializers\initializers_v2.py:120: UserWarning: The initializer RandomNormal is unseeded and being called multiple times, which will return identical values each time (even if the initializer is unseeded). Please update your code to provide a seed to the initializer, or avoid using the same initializer instance more than once.
```

```
    warnings.warn(
```

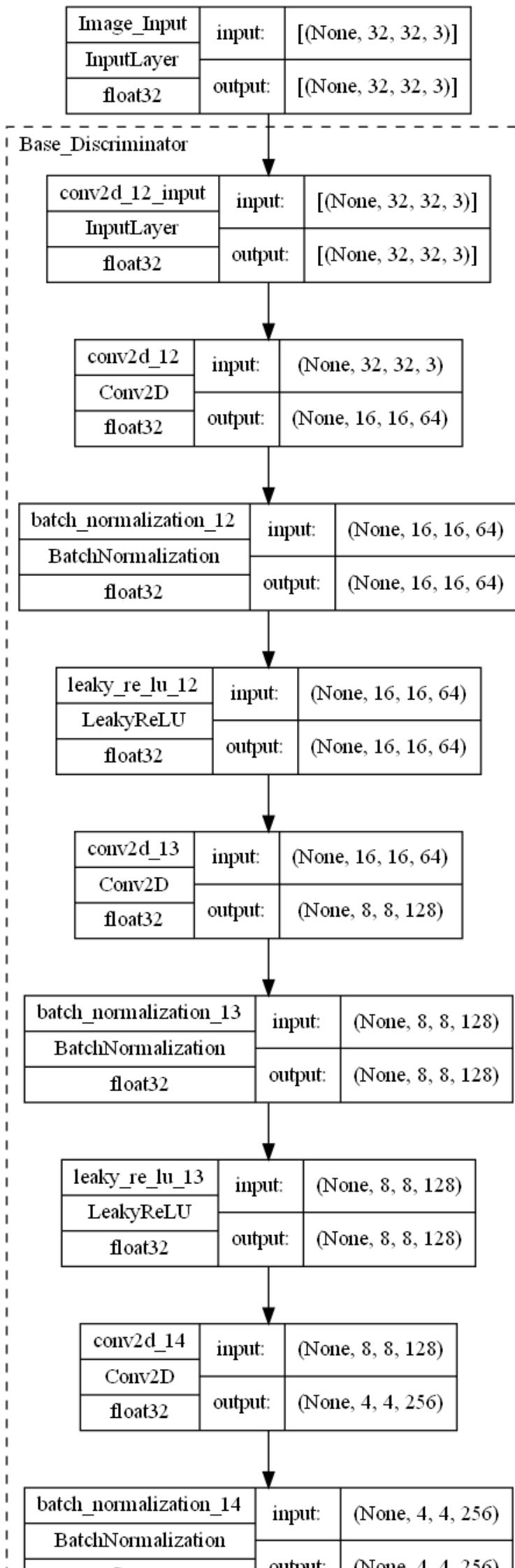
```
Model: "discriminator_ACGAN"
```

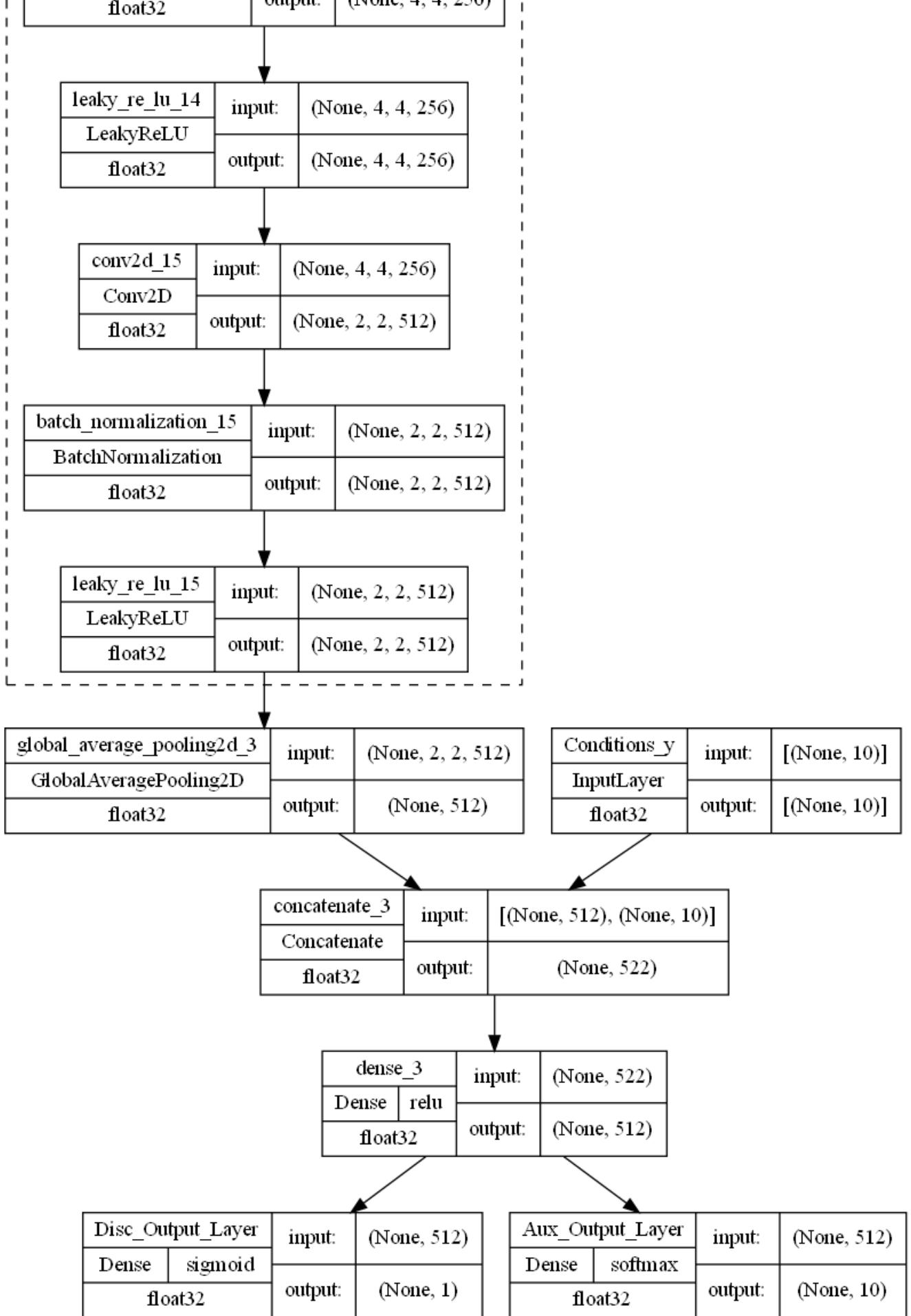
Layer (type)	Output Shape	Param #	Connected to
<hr/>			
=====			
Image_Input (InputLayer)	[None, 32, 32, 3]	0	[]
Base_Discriminator (Sequential)	(None, 2, 2, 512)	2760384	['Image_Input[0][0]']
<hr/>			
=====			
Image_Input (InputLayer)	[None, 32, 32, 3]	0	[]
Base_Discriminator (Sequential)	(None, 2, 2, 512)	2760384	['Image_Input[0][0]']
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0	['Base_Discriminator[0][0]']
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate (Concatenate)	(None, 522)	0	['global_average_pooling2d[0][0]', 'Conditions_y[0][0]']
dense (Dense)	(None, 512)	267776	['concatenate[0][0]']
Disc_Output_Layer (Dense)	(None, 1)	513	['dense[0][0]']
Aux_Output_Layer (Dense)	(None, 10)	5130	['dense[0][0]']
<hr/>			
=====			
Total params:	3,033,803		
Trainable params:	3,031,883		
Non-trainable params:	1,920		

```
In [ ]:
```

```
plot_model(create_improve_ACGAN_discriminator(image_size=IMG_SIZE), to_file="images/models/Ne  
show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True
```

Out[]:





New ACGAN Generator

Similarly, we will use Spectral Normalisation, Gaussian weights and PReLU for the generator. This will allow the generator to adapt better and learn better overall compared to the LeakyReLU used previously.

```
In [ ]: # function to create generator model
def create_improve_ACGAN_generator(noise):
```

```

# gaussian weights initialization
weights_init = RandomNormal(mean=0, stddev=0.02)

# latent noise vector z
z = Input(shape=(noise,), name="Latent_Noise_Vector_z")

# conditions y
conditions = Input(shape=(10,), name='Conditions_y')

# Generator network
merged_layer = Concatenate()([z, conditions])

# FC: 2x2x512
generator = Dense(2*2*512, activation='relu')(merged_layer)
generator = BatchNormalization(momentum=0.8)(generator)
generator = PReLU()(generator)
generator = Reshape((2, 2, 512))(generator)

base_generator = Sequential([
    # Conv 1: 4x4x256
    SpectralNormalization(Conv2DTranspose(256, kernel_size=4, strides=2,
                                         padding='same', kernel_initializer=weights_init),
                          BatchNormalization(momentum=0.8),
                          PReLU(),
    # Conv 2: 8x8x128
    SpectralNormalization(Conv2DTranspose(128, kernel_size=4, strides=2,
                                         padding='same', kernel_initializer=weights_init),
                          BatchNormalization(momentum=0.8),
                          PReLU(),
    # Conv 3: 16x16x64
    SpectralNormalization(Conv2DTranspose(64, kernel_size=4, strides=2,
                                         padding='same', kernel_initializer=weights_init),
                          BatchNormalization(momentum=0.8),
                          PReLU(),
], name='Base_Generator')
generator = base_generator(generator)

# Conv 4: 32x32x3
generator = Conv2DTranspose(3, kernel_size=4, strides=2, padding='same',
                           activation='tanh', name='Output_Layer')(generator)

generator = Model(inputs=[z, conditions],
                  outputs=generator, name='generator_ACGAN')
return generator

```

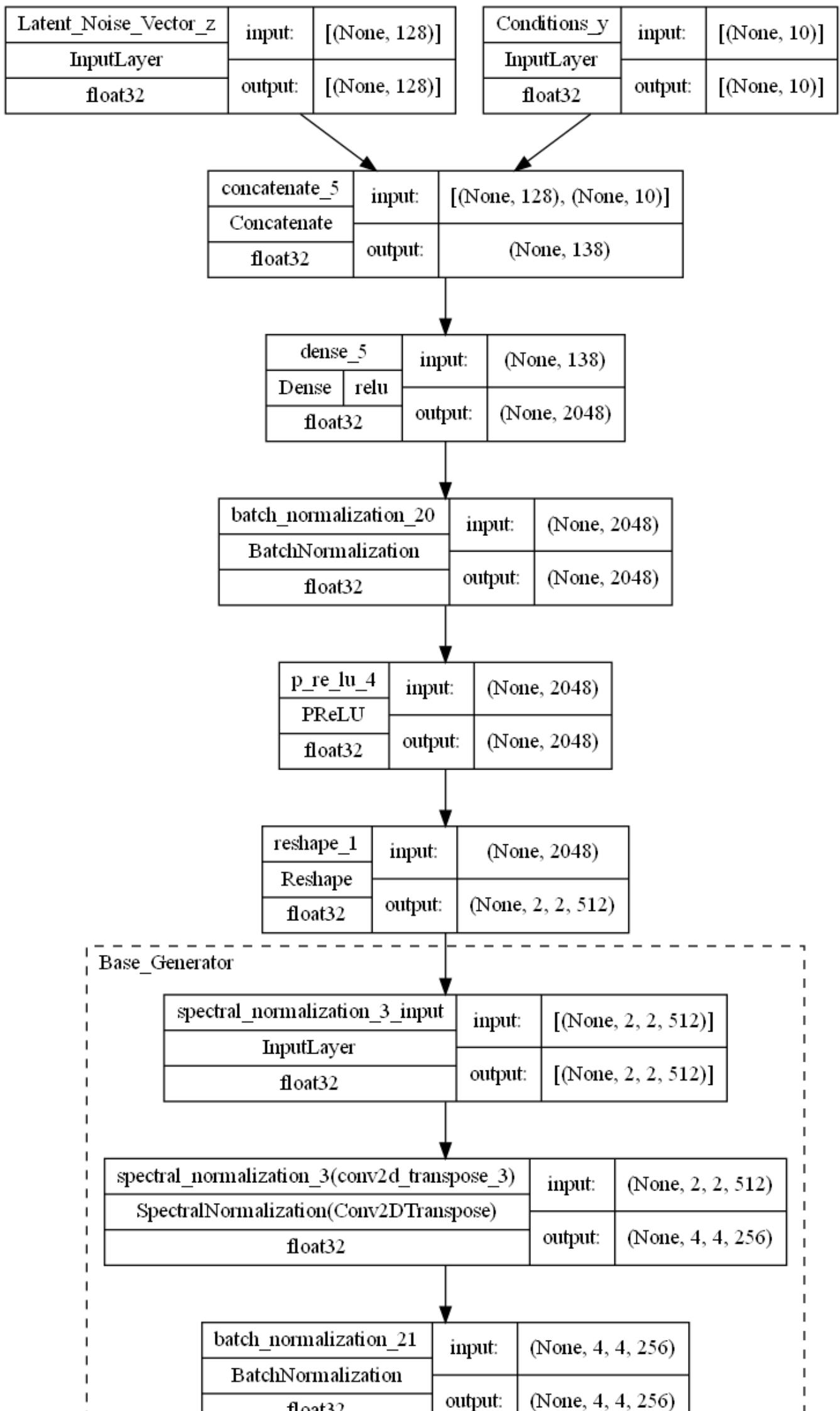
create_improve_ACGAN_generator(noise=NOISE).summary()

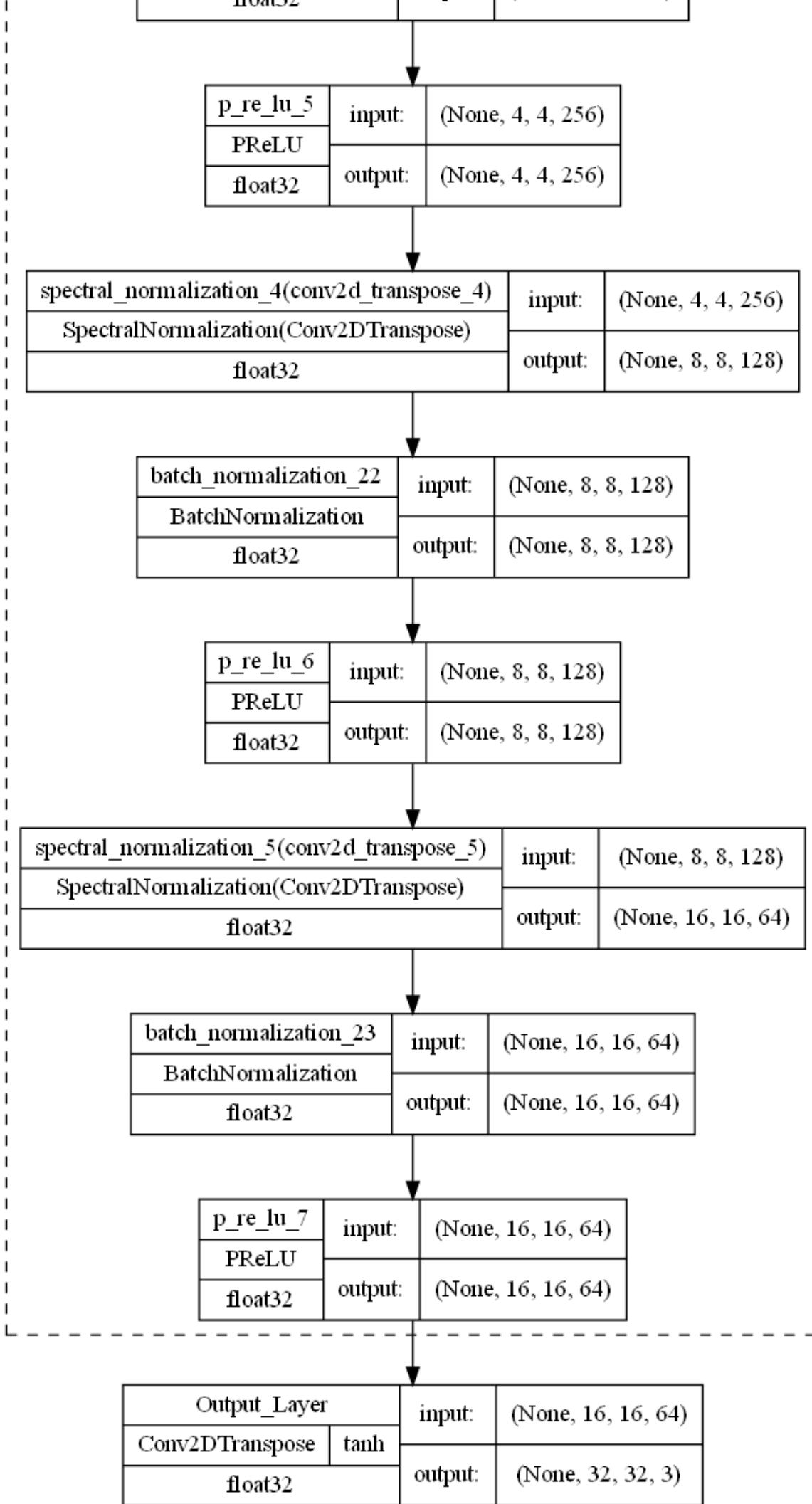
Model: "generator_ACGAN"

Layer (type)	Output Shape	Param #	Connected to
=====			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_1 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]',
=====			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_1 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_1 (Dense)	(None, 2048)	284672	['concatenate_1[0][0]']
batch_normalization_4 (BatchNormalization)	(None, 2048)	8192	['dense_1[0][0]']
p_re_lu (PReLU)	(None, 2048)	2048	['batch_normalization_4[0][0]']
reshape (Reshape)	(None, 2, 2, 512)	0	['p_re_lu[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2784320	['reshape[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
=====			
=====			
Total params: 3,082,307			
Trainable params: 3,076,419			
Non-trainable params: 5,888			

In []: `plot_model(create_improve_ACGAN_generator(noise=NOISE), to_file="images/models/New ACGAN gene show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True")`

Out[]:





```
In [ ]: class NewACGANMonitor(keras.callbacks.Callback):
    def __init__(self, num_img=20, noise=128, patience=10, vmin=0, vmax=1):
        self.num_img = num_img
        self.noise = noise
        self.patience = patience
        self.vmin = vmin
        self.vmax = vmax
        self.latent_noise_vector = tf.random.normal(
            shape=(self.num_img, self.noise))
        self.conditions = to_categorical([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                                         0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

    def generate_plot(self):
        # Generate Images
        generated_images = self.model.generator.predict(
            [self.latent_noise_vector, self.conditions])
        # Normalise Image from [vmin, vmax] to [0, 1]
        generated_images -= self.vmin
        generated_images /= (self.vmax - self.vmin)
        row_size = int(np.ceil(self.num_img/5))
        fig = plt.figure(figsize=(10, 2*row_size), tight_layout=True)
        for i in range(self.num_img):
            ax = fig.add_subplot(row_size, 5, i+1)
            ax.imshow(generated_images[i])
            ax.set_title(class_labels[i % 10])
            ax.axis('off')
        plt.show()

    def save_weights(self, epoch=None):
        try:
            if epoch != None:
                name = 'New ACGAN/generator-epoch-{}.h5'.format(epoch)
                print('Generator Checkpoint - {}'.format(name))
                self.model.generator.save_weights(
                    filepath=name,
                    save_format='h5'
                )
        except Exception as e:
            print(e)

    def on_epoch_begin(self, epoch, logs=None):
        if epoch % self.patience == 0:
            self.generate_plot()
            self.save_weights(epoch)

    def on_train_end(self, epoch, logs=None):
        self.generate_plot()
        self.save_weights('Full Train')
```

```
In [ ]: callbacks = [
    NewACGANMonitor(num_img=20, noise=128, patience=5, vmin=-1, vmax=1),
]
```

New ACGAN Training

```
In [ ]: class AConditionalGAN(tf.keras.Model):
    def __init__(self, discriminator, generator, noise):
        super(AConditionalGAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.noise = noise
        self.gen_loss_tracker = tf.keras.metrics.Mean(name="generator_loss")
        self.disc_loss_tracker = tf.keras.metrics.Mean(
            name="discriminator_loss")
```

```

self.d_xy_tracker = tf.keras.metrics.Mean(name='Mean D(x|y)')
self.d_g_zy_tracker = tf.keras.metrics.Mean(name='Mean D(G(z|y))')
self.kl = tf.keras.metrics.KLDivergence()

def compile(self, d_optimizer, g_optimizer, disc_loss_fn, aux_loss_fn):
    super(AConditionalGAN, self).compile()
    self.d_optimizer = d_optimizer
    self.g_optimizer = g_optimizer
    self.disc_loss_fn = disc_loss_fn
    self.aux_loss_fn = aux_loss_fn

def train_step(self, data):
    ### TRAINING DISCRIMINATOR ###
    # Unpack the data.
    real_images, condition = data

    # Sample for latent noise vector z
    batch_size = tf.shape(real_images)[0]
    latent_noise_vector = tf.random.normal(
        shape=(batch_size, self.noise))

    # Maps the noise Latent vector and Labels to generate fake images.
    generated_images = self.generator([latent_noise_vector, condition])

    # Combine with real images
    combined_images = tf.concat([generated_images, real_images], axis=0)
    combined_condition = tf.concat([condition, condition], axis=0)

    # Discrimination
    labels = tf.concat(
        [tf.zeros((batch_size, 1)), tf.ones((batch_size, 1))], axis=0
    )

    # Train the discriminator.
    with tf.GradientTape() as tape:
        disc_output, aux_output = self.discriminator(
            [combined_images, combined_condition])
        disc_d_loss = self.disc_loss_fn(labels, disc_output)
        aux_d_loss = self.aux_loss_fn(combined_condition, aux_output)
        d_loss = disc_d_loss + aux_d_loss
        grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights)
        )

    # Computing D(x/y)
    d_xy = tf.math.reduce_mean(disc_output)

    ### TRAINING GENERATOR ###
    latent_noise_vector = tf.random.normal(
        shape=(batch_size, self.noise))

    # Assemble Labels that say "all real images".
    misleading_labels = tf.ones((batch_size, 1))

    # Train the Generator
    with tf.GradientTape() as tape:
        fake_images = self.generator([latent_noise_vector, condition])
        disc_output, aux_output = self.discriminator(
            [fake_images, condition])
        disc_g_loss = self.disc_loss_fn(misleading_labels, disc_output)
        aux_g_loss = self.aux_loss_fn(condition, aux_output)
        g_loss = disc_g_loss + aux_g_loss
        grads = tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(
            zip(grads, self.generator.trainable_weights))

```

```

# Computing D(G(z|y))
d_g_zy = tf.math.reduce_mean(disc_output)

# Monitor loss and metrics.
self.gen_loss_tracker.update_state(g_loss)
self.disc_loss_tracker.update_state(d_loss)
self.d_xy_tracker.update_state(d_xy)
self.d_g_zy_tracker.update_state(d_g_zy)
self.kl.update_state(real_images, fake_images)

return {
    "d_loss": self.disc_loss_tracker.result(),
    "g_loss": self.gen_loss_tracker.result(),
    "D(x|y)": self.d_xy_tracker.result(),
    "D(G(z|y))": self.d_g_zy_tracker.result(),
    "KL Divergence": self.kl.result(),
}

```

```

In [ ]: tf.keras.backend.clear_session()
K.clear_session()

improve_aux_cond_gan = AConditionalGAN(
    discriminator=create_improve_ACGAN_discriminator(image_size=IMG_SIZE),
    generator=create_improve_ACGAN_generator(noise=NOISE),
    noise=NOISE
)

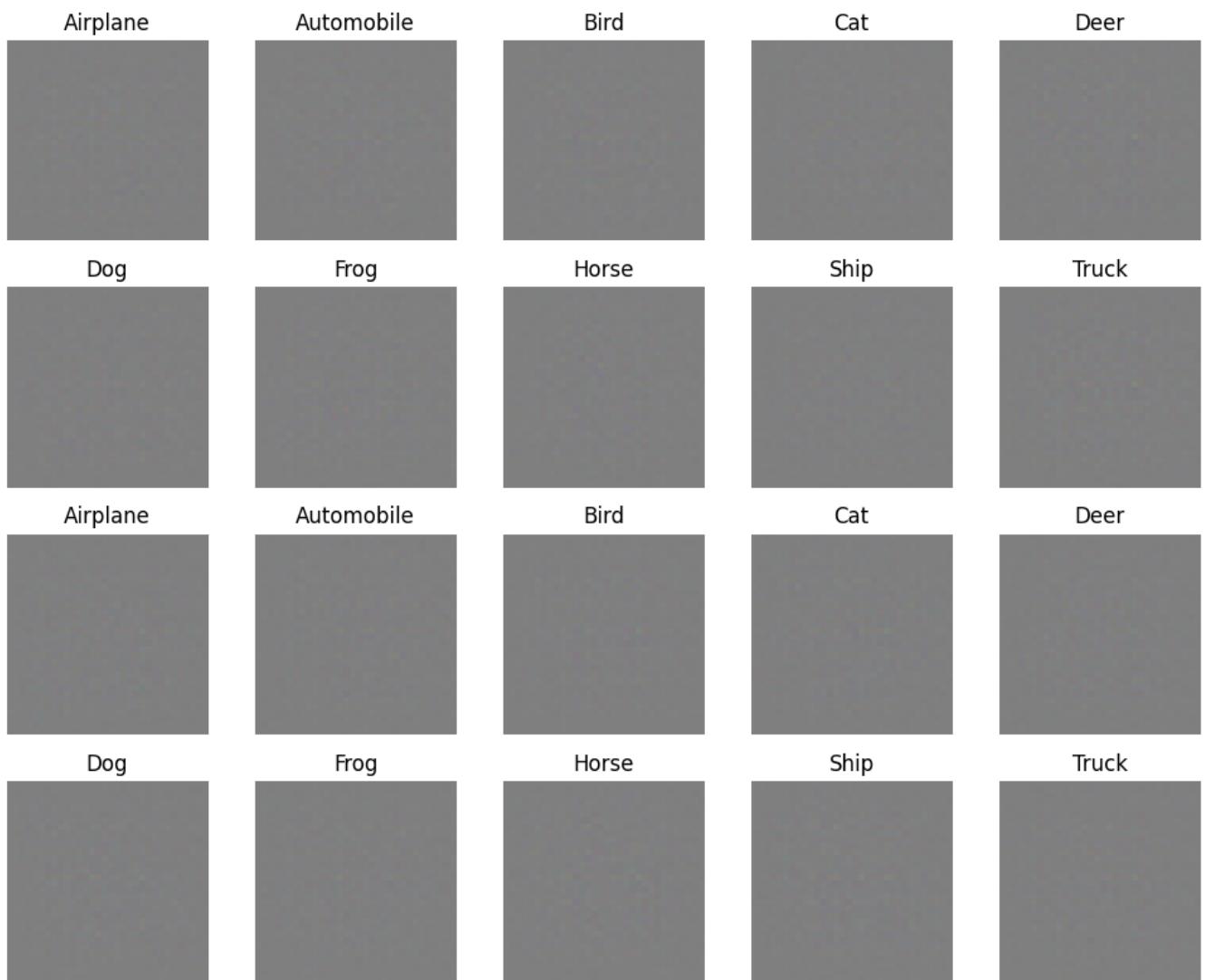
improve_aux_cond_gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    disc_loss_fn=keras.losses.BinaryCrossentropy(),
    aux_loss_fn=keras.losses.CategoricalCrossentropy()
)

dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
dataset = dataset.shuffle(buffer_size=1024).batch(
    BATCH_SIZE, num_parallel_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)

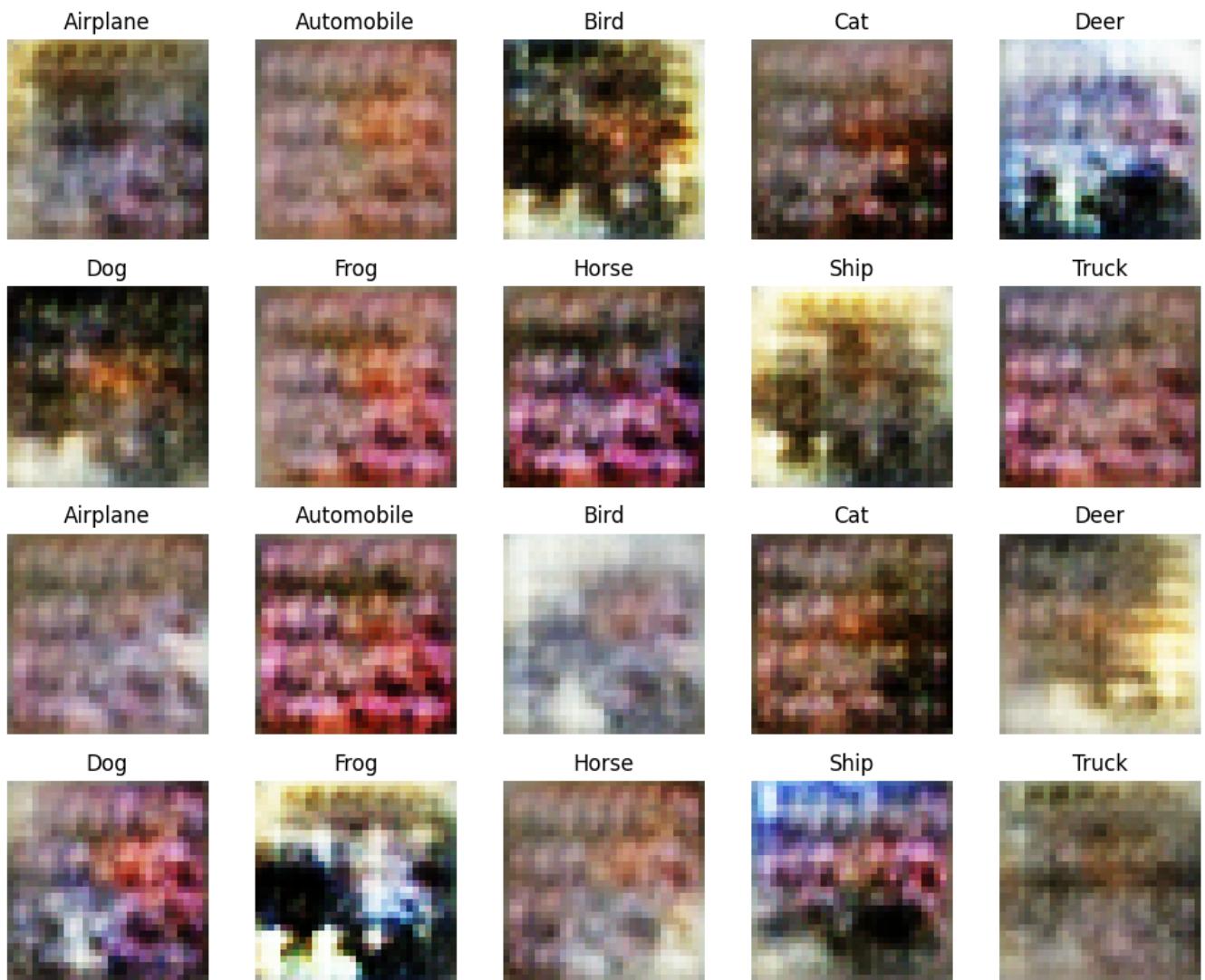
improve_aux_cond_hist = improve_aux_cond_gan.fit(
    dataset, epochs=200, use_multiprocessing=True, workers=16, callbacks=callbacks)

```

1/1 [=====] - 4s 4s/step



Generator Checkpoint - New ACGAN/generator-epoch-0.h5
Epoch 1/200
782/782 [=====] - 109s 87ms/step - d_loss: 1.6439 - g_loss: 4.4664 -
D(x|y): 0.5262 - D(G(z|y)): 0.1249 - KL Divergence: 5.0581
Epoch 2/200
782/782 [=====] - 88s 112ms/step - d_loss: 0.3334 - g_loss: 3.3972 -
D(x|y): 0.4955 - D(G(z|y)): 0.0696 - KL Divergence: 5.6292
Epoch 3/200
782/782 [=====] - 82s 105ms/step - d_loss: 0.3269 - g_loss: 3.3939 -
D(x|y): 0.4955 - D(G(z|y)): 0.0747 - KL Divergence: 4.8018
Epoch 4/200
782/782 [=====] - 96s 123ms/step - d_loss: 0.2873 - g_loss: 2.9727 -
D(x|y): 0.5010 - D(G(z|y)): 0.0848 - KL Divergence: 5.2614
Epoch 5/200
782/782 [=====] - 312s 399ms/step - d_loss: 0.2686 - g_loss: 3.1902 -
D(x|y): 0.5025 - D(G(z|y)): 0.0723 - KL Divergence: 4.7380
1/1 [=====] - 0s 40ms/step



Generator Checkpoint - New ACGAN/generator-epoch-5.h5

Epoch 6/200

782/782 [=====] - 71s 91ms/step - d_loss: 0.2766 - g_loss: 3.1532 -
 $D(x|y)$: 0.5018 - $D(G(z|y))$: 0.0753 - KL Divergence: 5.1060

Epoch 7/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.2999 - g_loss: 2.9464 -
 $D(x|y)$: 0.5010 - $D(G(z|y))$: 0.1009 - KL Divergence: 5.0076

Epoch 8/200

782/782 [=====] - 77s 99ms/step - d_loss: 0.2905 - g_loss: 2.9631 -
 $D(x|y)$: 0.4994 - $D(G(z|y))$: 0.1016 - KL Divergence: 4.5838

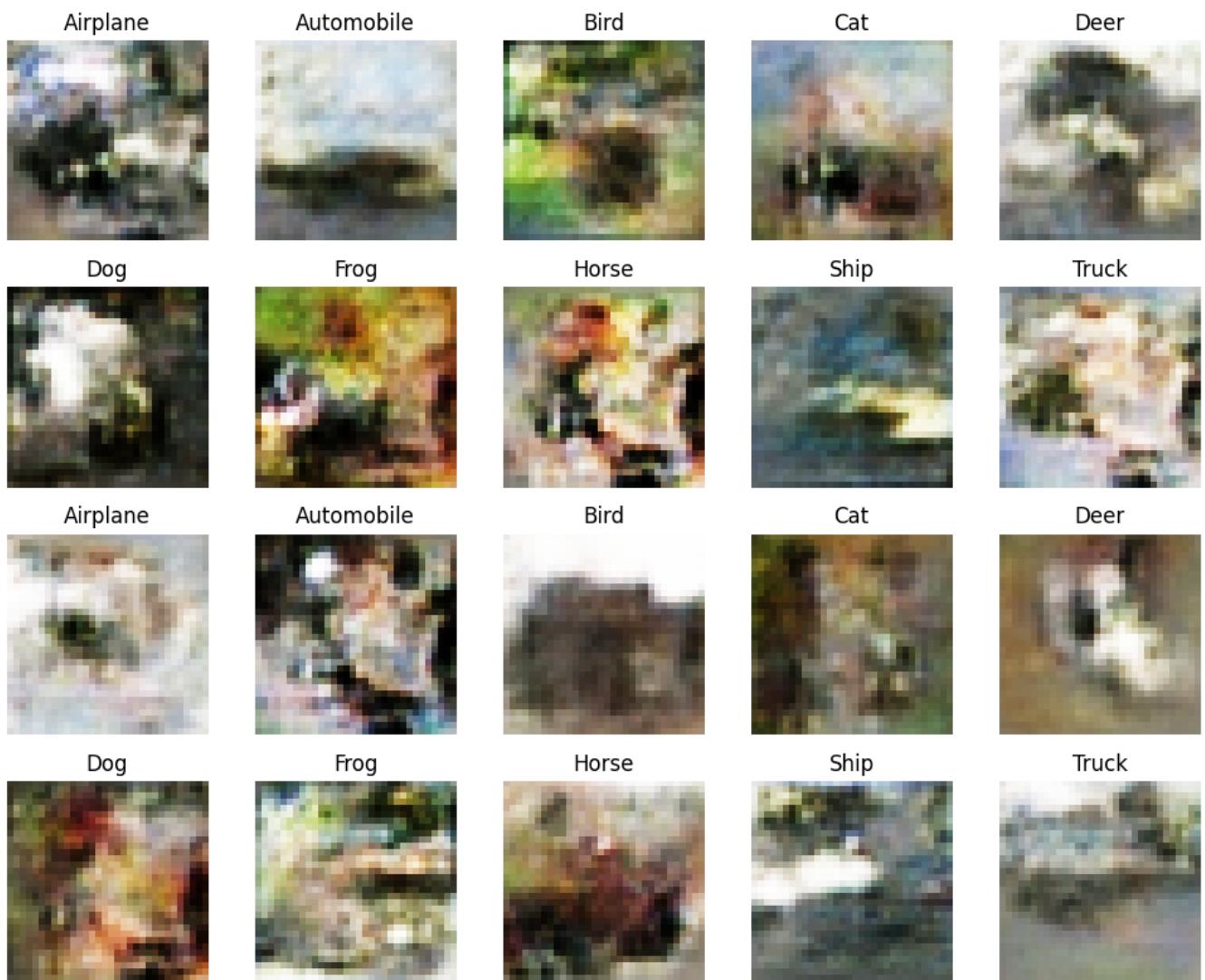
Epoch 9/200

782/782 [=====] - 73s 93ms/step - d_loss: 0.2880 - g_loss: 2.9228 -
 $D(x|y)$: 0.5009 - $D(G(z|y))$: 0.1095 - KL Divergence: 4.5984

Epoch 10/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.2843 - g_loss: 2.8786 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.1158 - KL Divergence: 4.6413

1/1 [=====] - 0s 41ms/step



Generator Checkpoint - New ACGAN/generator-epoch-10.h5

Epoch 11/200

782/782 [=====] - 73s 93ms/step - d_loss: 0.2865 - g_loss: 2.8325 -
 $D(x|y): 0.5019$ - $D(G(z|y)): 0.1232$ - KL Divergence: 4.5089

Epoch 12/200

782/782 [=====] - 72s 93ms/step - d_loss: 0.3141 - g_loss: 2.6075 -
 $D(x|y): 0.4996$ - $D(G(z|y)): 0.1471$ - KL Divergence: 4.4697

Epoch 13/200

782/782 [=====] - 71s 91ms/step - d_loss: 0.3046 - g_loss: 2.6422 -
 $D(x|y): 0.5013$ - $D(G(z|y)): 0.1438$ - KL Divergence: 4.6338

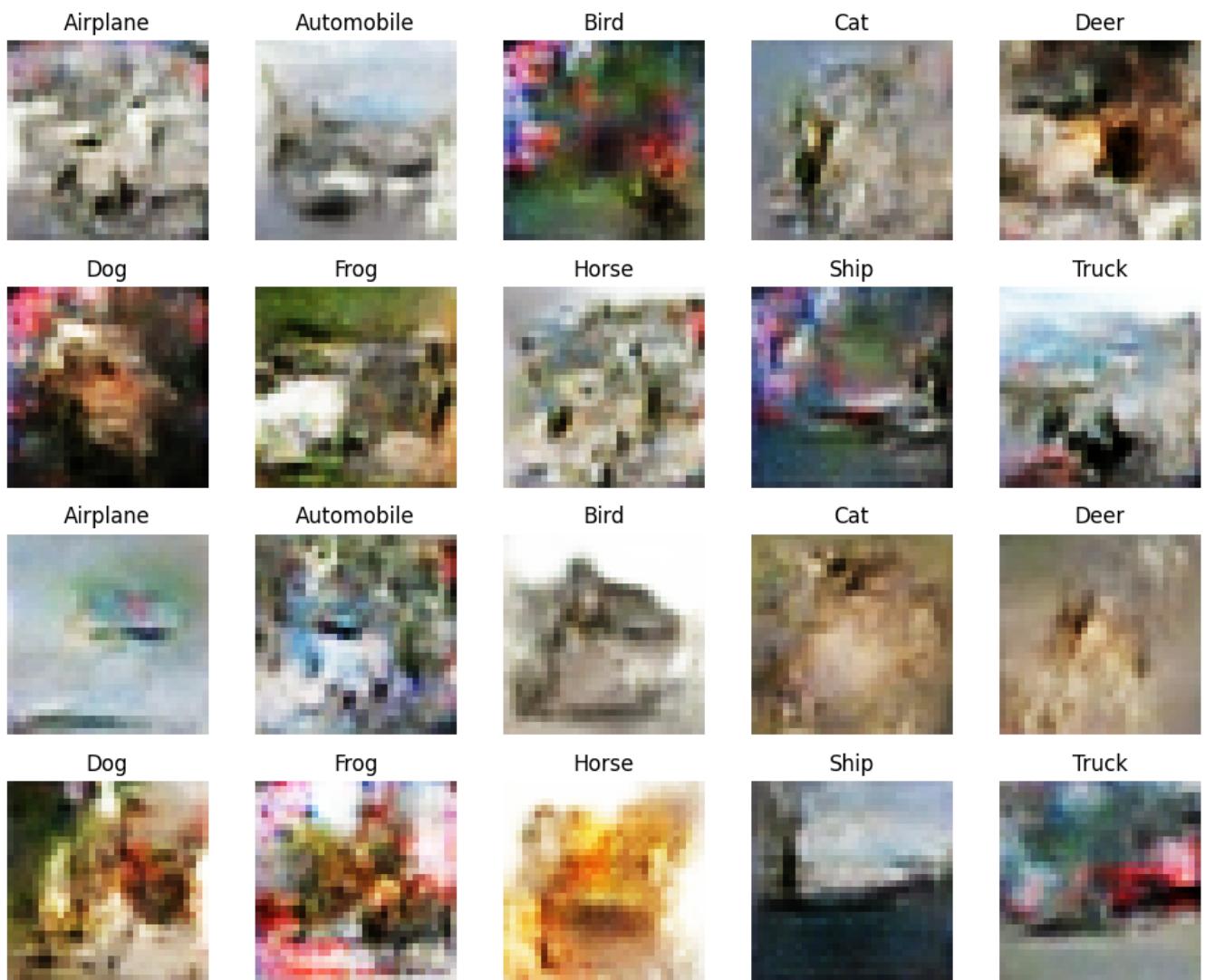
Epoch 14/200

782/782 [=====] - 196s 250ms/step - d_loss: 0.3209 - g_loss: 2.5556
 $- D(x|y): 0.5005$ - $D(G(z|y)): 0.1546$ - KL Divergence: 4.5383

Epoch 15/200

782/782 [=====] - 72s 92ms/step - d_loss: 0.3140 - g_loss: 2.5321 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.1566$ - KL Divergence: 4.5920

1/1 [=====] - 0s 46ms/step



Generator Checkpoint - New ACGAN/generator-epoch-15.h5

Epoch 16/200

782/782 [=====] - 80s 102ms/step - d_loss: 0.3165 - g_loss: 2.4437 -
 $D(x|y)$: 0.5009 - $D(G(z|y))$: 0.1664 - KL Divergence: 4.6190

Epoch 17/200

782/782 [=====] - 78s 99ms/step - d_loss: 0.3044 - g_loss: 2.5265 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.1592 - KL Divergence: 4.5491

Epoch 18/200

782/782 [=====] - 92s 117ms/step - d_loss: 0.3748 - g_loss: 2.0873 -
 $D(x|y)$: 0.4971 - $D(G(z|y))$: 0.2152 - KL Divergence: 4.6130

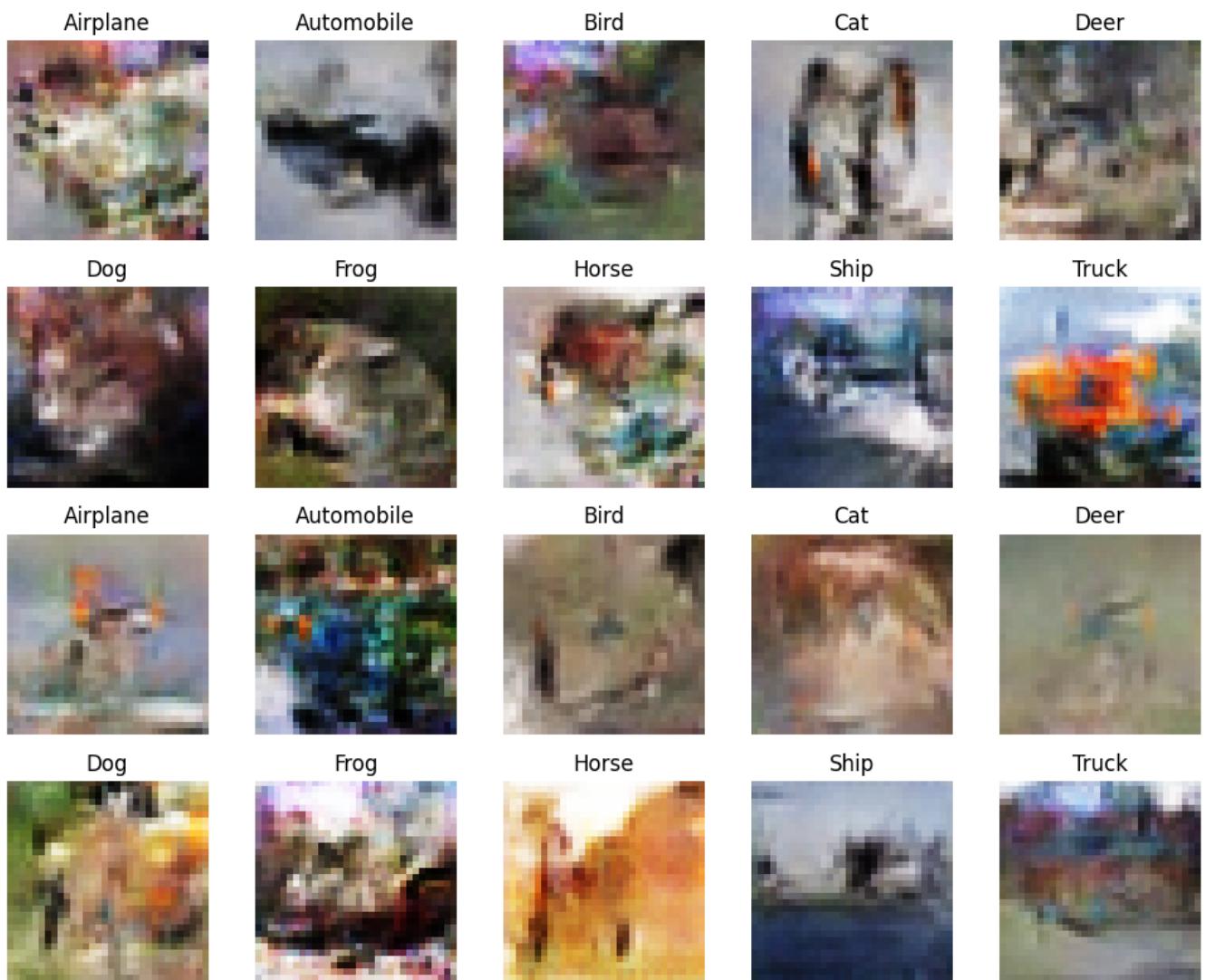
Epoch 19/200

782/782 [=====] - 74s 95ms/step - d_loss: 0.3095 - g_loss: 2.5011 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.1647 - KL Divergence: 4.5178

Epoch 20/200

782/782 [=====] - 152s 194ms/step - d_loss: 0.2898 - g_loss: 2.5939 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.1593 - KL Divergence: 4.6552

1/1 [=====] - 0s 98ms/step



Generator Checkpoint - New ACGAN/generator-epoch-20.h5

Epoch 21/200

782/782 [=====] - 75s 95ms/step - d_loss: 0.2879 - g_loss: 2.6576 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.1550 - KL Divergence: 4.6697

Epoch 22/200

782/782 [=====] - 116s 148ms/step - d_loss: 0.2856 - g_loss: 2.6794
 $- D(x|y)$: 0.5009 - $D(G(z|y))$: 0.1553 - KL Divergence: 4.7003

Epoch 23/200

782/782 [=====] - 106s 135ms/step - d_loss: 0.2739 - g_loss: 2.7775
 $- D(x|y)$: 0.5005 - $D(G(z|y))$: 0.1459 - KL Divergence: 4.5943

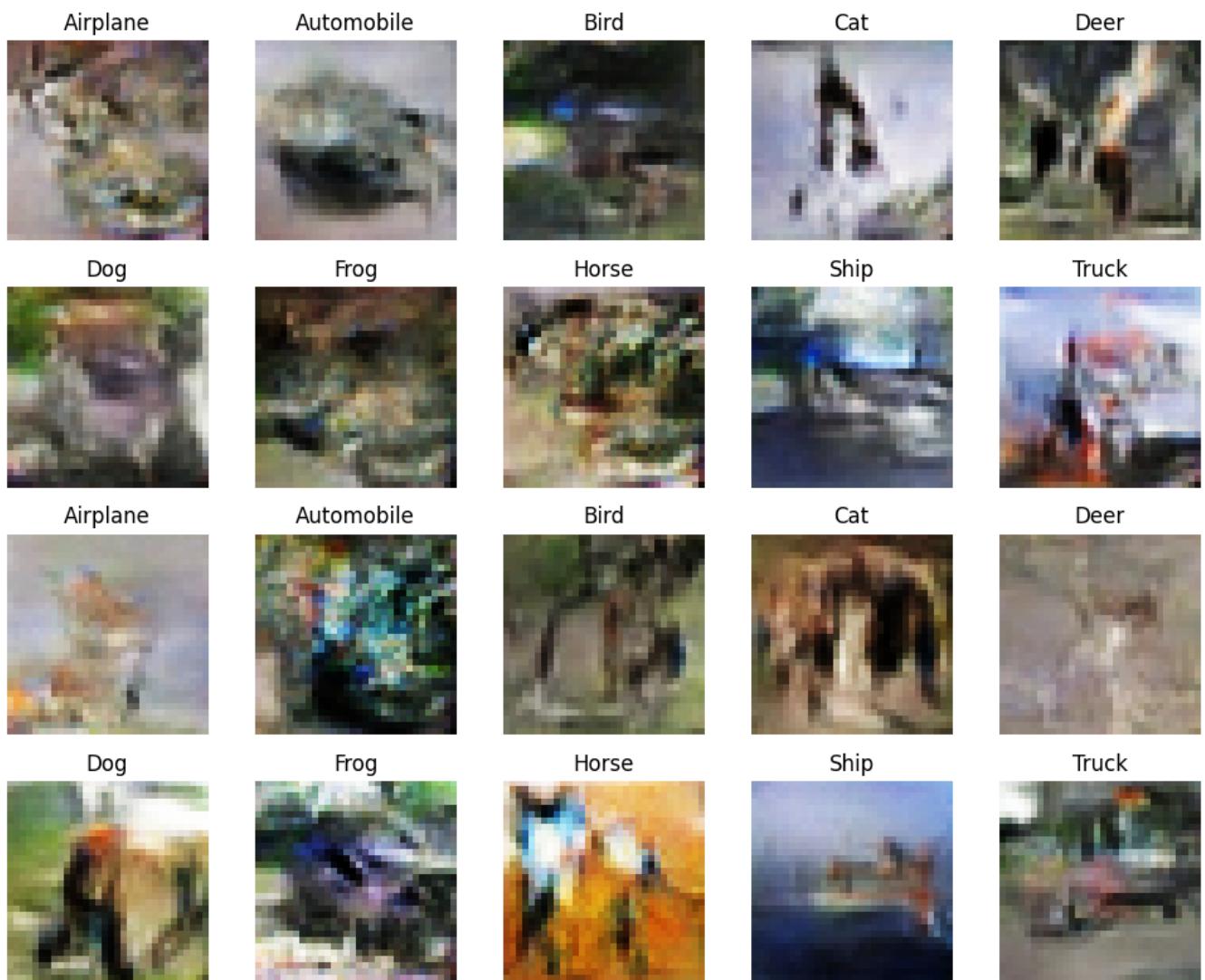
Epoch 24/200

782/782 [=====] - 83s 107ms/step - d_loss: 0.2707 - g_loss: 2.8058 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.1473 - KL Divergence: 4.6499

Epoch 25/200

782/782 [=====] - 96s 122ms/step - d_loss: 0.2928 - g_loss: 2.7117 -
 $D(x|y)$: 0.4992 - $D(G(z|y))$: 0.1633 - KL Divergence: 4.6439

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New ACGAN/generator-epoch-25.h5

Epoch 26/200

782/782 [=====] - 96s 122ms/step - d_loss: 0.2652 - g_loss: 2.8549 -
 $D(x|y)$: 0.5009 - $D(G(z|y))$: 0.1479 - KL Divergence: 4.8003

Epoch 27/200

782/782 [=====] - 90s 116ms/step - d_loss: 0.2600 - g_loss: 2.8982 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.1448 - KL Divergence: 4.7303

Epoch 28/200

782/782 [=====] - 101s 130ms/step - d_loss: 0.2643 - g_loss: 2.8203
 $- D(x|y)$: 0.5005 - $D(G(z|y))$: 0.1520 - KL Divergence: 4.7508

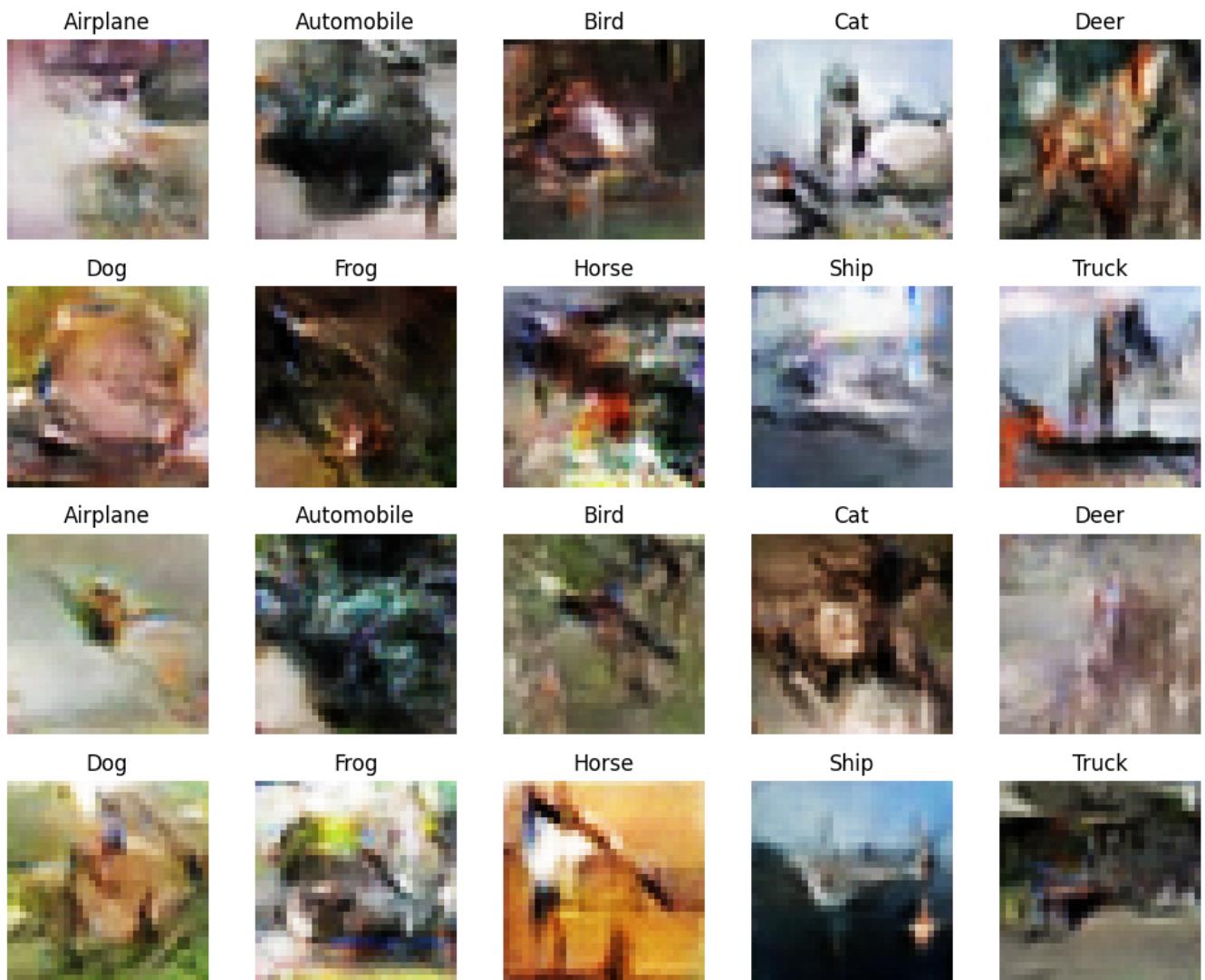
Epoch 29/200

782/782 [=====] - 98s 125ms/step - d_loss: 0.2583 - g_loss: 2.9410 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.1457 - KL Divergence: 4.7254

Epoch 30/200

782/782 [=====] - 99s 126ms/step - d_loss: 0.2451 - g_loss: 3.0769 -
 $D(x|y)$: 0.4996 - $D(G(z|y))$: 0.1347 - KL Divergence: 4.6875

1/1 [=====] - 0s 33ms/step



Generator Checkpoint - New ACGAN/generator-epoch-30.h5

Epoch 31/200

782/782 [=====] - 97s 124ms/step - d_loss: 0.2351 - g_loss: 3.1060 - D(x|y): 0.4995 - D(G(z|y)): 0.1324 - KL Divergence: 4.7472

Epoch 32/200

782/782 [=====] - 99s 126ms/step - d_loss: 0.2239 - g_loss: 3.3110 - D(x|y): 0.5002 - D(G(z|y)): 0.1253 - KL Divergence: 4.7180

Epoch 33/200

782/782 [=====] - 108s 138ms/step - d_loss: 0.2178 - g_loss: 3.3819 - D(x|y): 0.5003 - D(G(z|y)): 0.1259 - KL Divergence: 4.7481

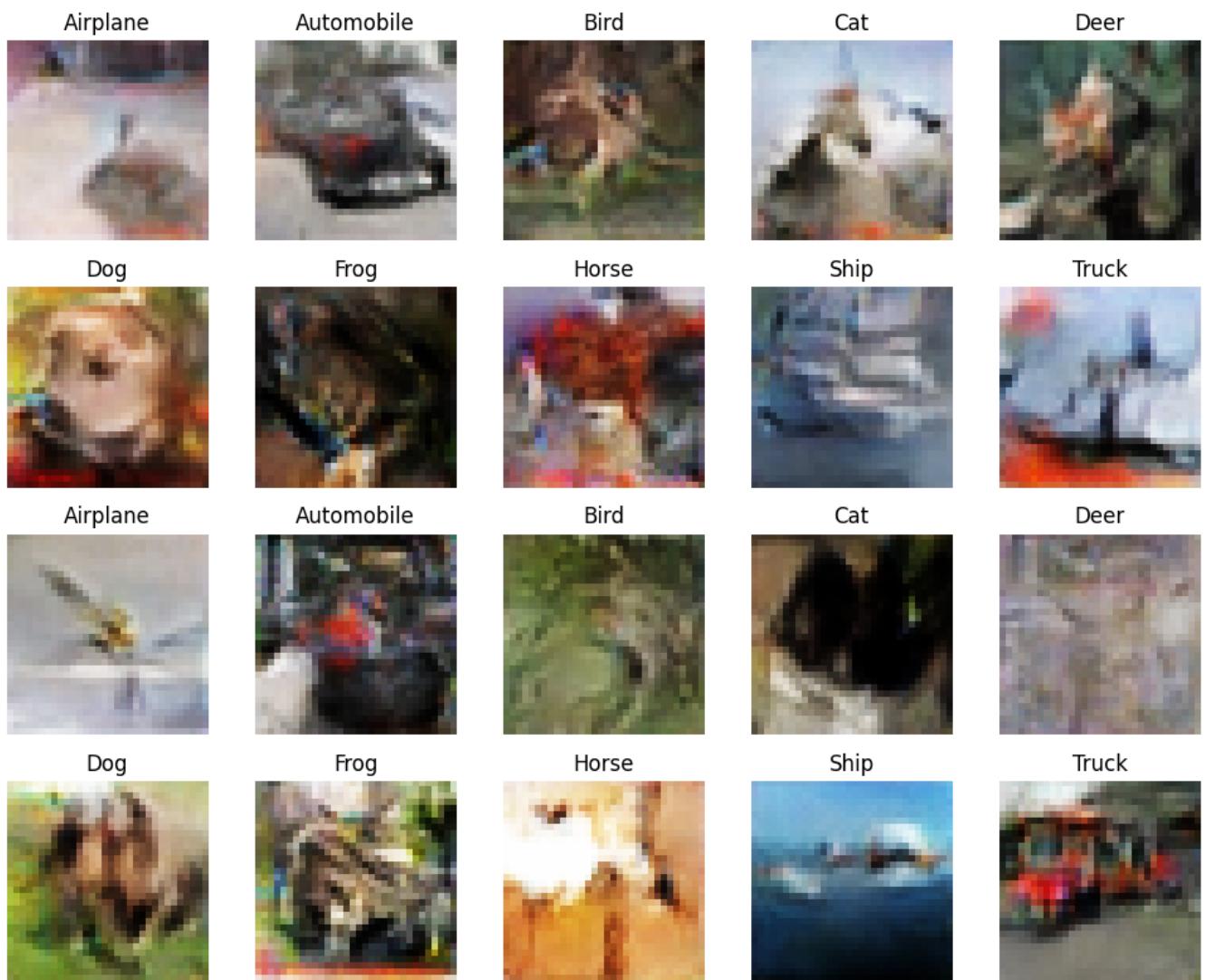
Epoch 34/200

782/782 [=====] - 99s 127ms/step - d_loss: 0.2132 - g_loss: 3.4759 - D(x|y): 0.5011 - D(G(z|y)): 0.1176 - KL Divergence: 4.7260

Epoch 35/200

782/782 [=====] - 102s 131ms/step - d_loss: 0.2199 - g_loss: 3.4642 - D(x|y): 0.5005 - D(G(z|y)): 0.1236 - KL Divergence: 4.7087

1/1 [=====] - 0s 30ms/step



Generator Checkpoint - New ACGAN/generator-epoch-35.h5

Epoch 36/200

```
782/782 [=====] - 112s 143ms/step - d_loss: 0.2061 - g_loss: 3.5923
- D(x|y): 0.5002 - D(G(z|y)): 0.1141 - KL Divergence: 4.7718
```

Epoch 37/200

```
782/782 [=====] - 95s 122ms/step - d_loss: 0.2149 - g_loss: 3.6089
- D(x|y): 0.5005 - D(G(z|y)): 0.1164 - KL Divergence: 4.7359
```

Epoch 38/200

```
782/782 [=====] - 96s 122ms/step - d_loss: 0.1958 - g_loss: 3.7690
- D(x|y): 0.5003 - D(G(z|y)): 0.1095 - KL Divergence: 4.8173
```

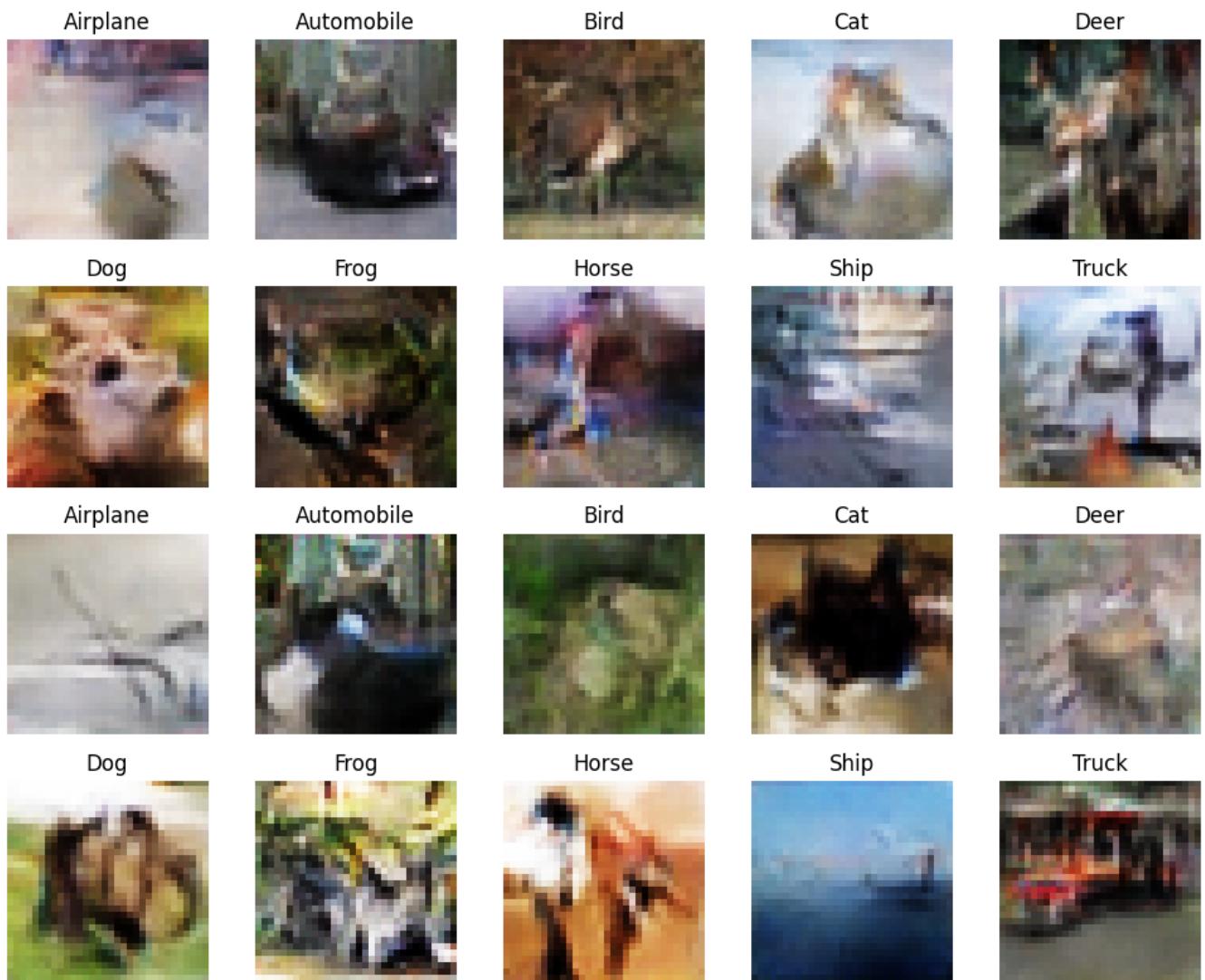
Epoch 39/200

```
782/782 [=====] - 85s 108ms/step - d_loss: 0.1909 - g_loss: 3.8559
- D(x|y): 0.5003 - D(G(z|y)): 0.1082 - KL Divergence: 4.7473
```

Epoch 40/200

```
782/782 [=====] - 84s 107ms/step - d_loss: 0.1929 - g_loss: 3.9237
- D(x|y): 0.5001 - D(G(z|y)): 0.1063 - KL Divergence: 4.8382
```

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-40.h5

Epoch 41/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.3209 - g_loss: 3.3620 - D(x|y): 0.5002 - D(G(z|y)): 0.1494 - KL Divergence: 4.7377

Epoch 42/200

782/782 [=====] - 84s 107ms/step - d_loss: 0.1911 - g_loss: 3.9264 - D(x|y): 0.5009 - D(G(z|y)): 0.1072 - KL Divergence: 4.6980

Epoch 43/200

782/782 [=====] - 87s 111ms/step - d_loss: 0.1885 - g_loss: 3.9951 - D(x|y): 0.5007 - D(G(z|y)): 0.1031 - KL Divergence: 4.7631

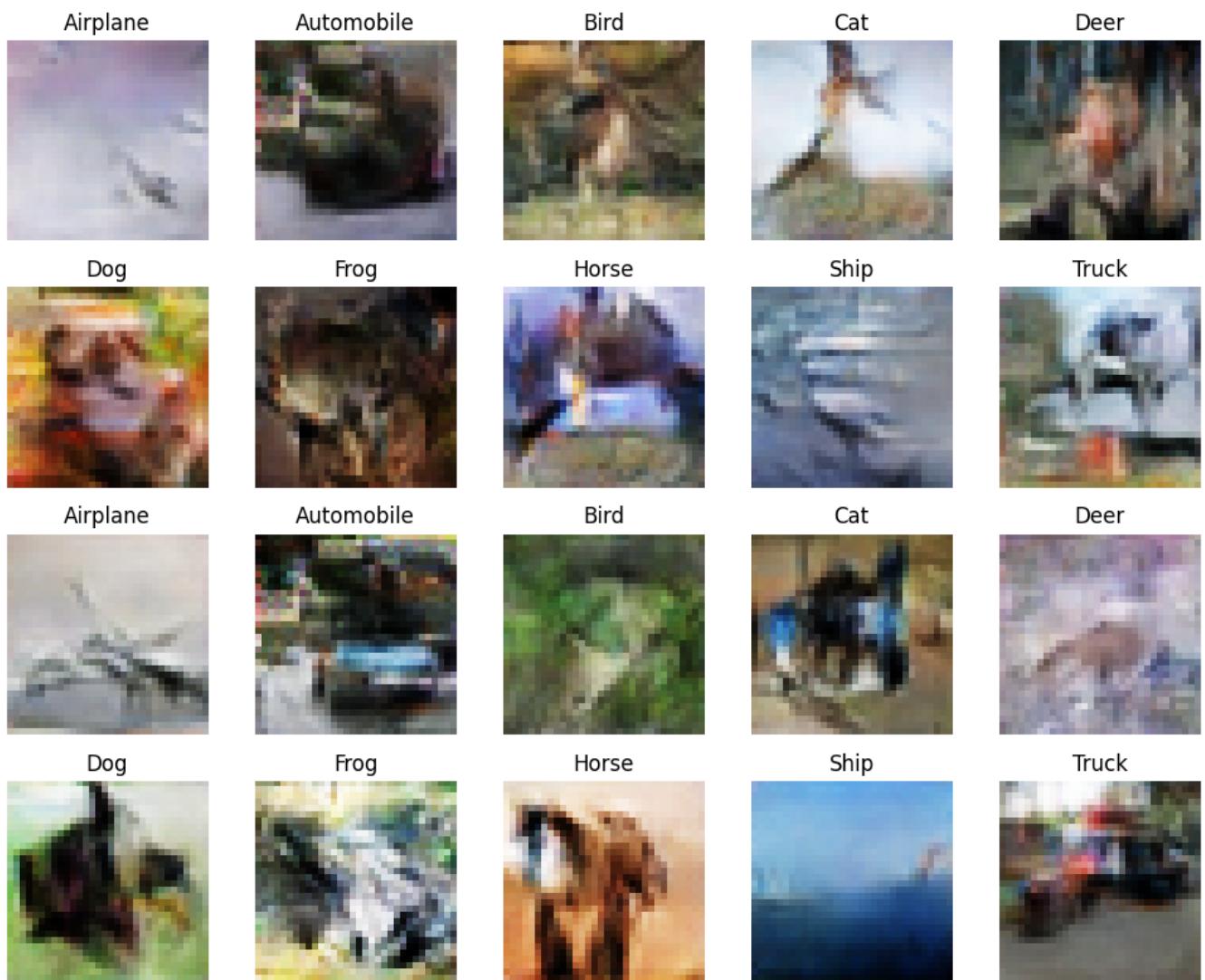
Epoch 44/200

782/782 [=====] - 83s 106ms/step - d_loss: 0.1847 - g_loss: 4.1100 - D(x|y): 0.5004 - D(G(z|y)): 0.1004 - KL Divergence: 4.8429

Epoch 45/200

782/782 [=====] - 83s 106ms/step - d_loss: 0.1883 - g_loss: 4.0853 - D(x|y): 0.5006 - D(G(z|y)): 0.1037 - KL Divergence: 4.7726

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New ACGAN/generator-epoch-45.h5

Epoch 46/200

782/782 [=====] - 87s 111ms/step - d_loss: 0.1819 - g_loss: 4.1744 - D(x|y): 0.4999 - D(G(z|y)): 0.1005 - KL Divergence: 4.7909

Epoch 47/200

782/782 [=====] - 83s 106ms/step - d_loss: 0.1773 - g_loss: 4.2348 - D(x|y): 0.5002 - D(G(z|y)): 0.1001 - KL Divergence: 4.7021

Epoch 48/200

782/782 [=====] - 83s 106ms/step - d_loss: 0.1705 - g_loss: 4.4048 - D(x|y): 0.5002 - D(G(z|y)): 0.0935 - KL Divergence: 4.7863

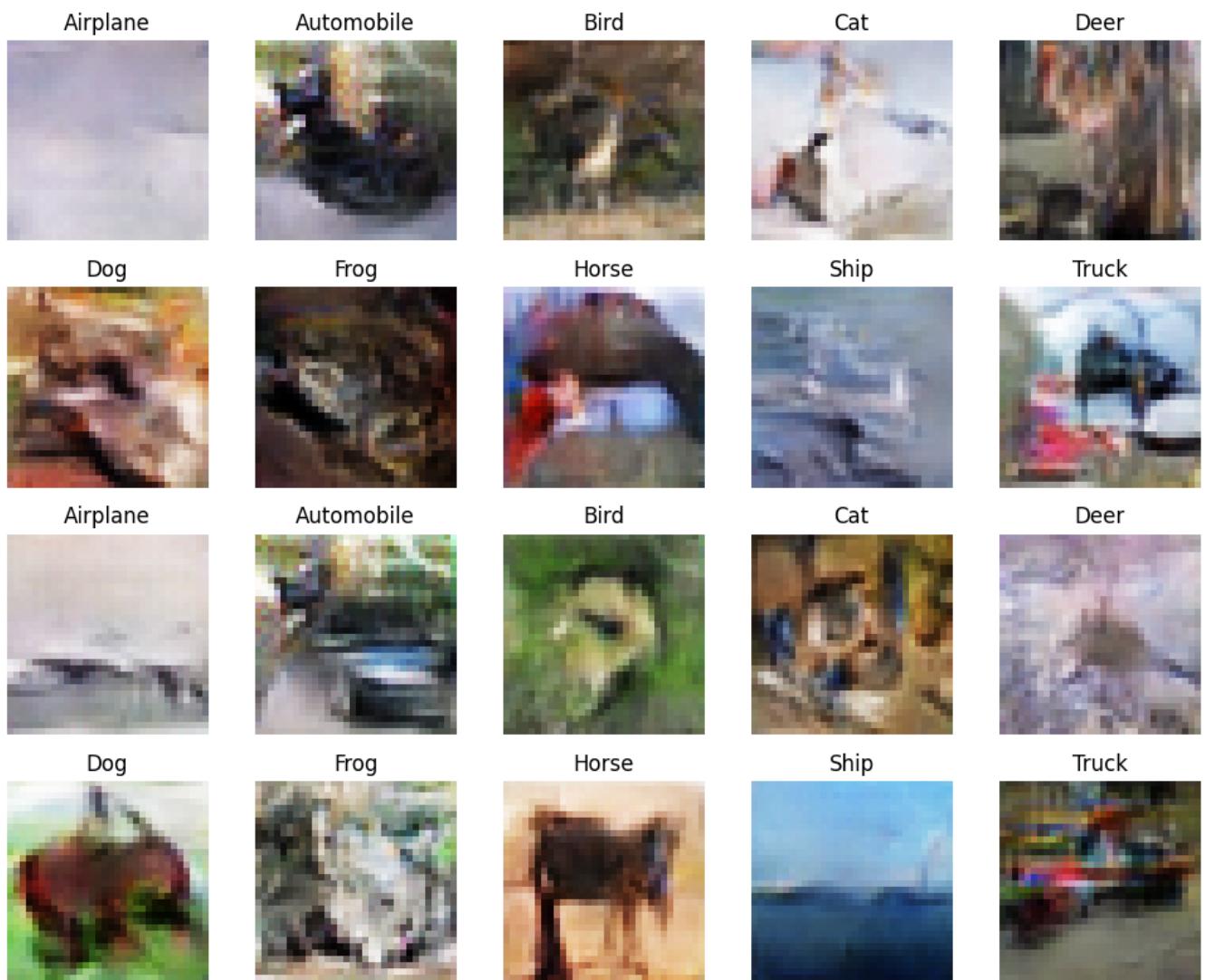
Epoch 49/200

782/782 [=====] - 87s 112ms/step - d_loss: 0.1743 - g_loss: 4.4208 - D(x|y): 0.5007 - D(G(z|y)): 0.0960 - KL Divergence: 4.7784

Epoch 50/200

782/782 [=====] - 84s 107ms/step - d_loss: 0.1630 - g_loss: 4.4945 - D(x|y): 0.5003 - D(G(z|y)): 0.0915 - KL Divergence: 4.7848

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New ACGAN/generator-epoch-50.h5

Epoch 51/200

```
782/782 [=====] - 87s 111ms/step - d_loss: 0.1654 - g_loss: 4.5334 -
D(x|y): 0.5009 - D(G(z|y)): 0.0925 - KL Divergence: 4.7669
```

Epoch 52/200

```
782/782 [=====] - 87s 111ms/step - d_loss: 0.1678 - g_loss: 4.5499 -
D(x|y): 0.5003 - D(G(z|y)): 0.0926 - KL Divergence: 4.5998
```

Epoch 53/200

```
782/782 [=====] - 83s 106ms/step - d_loss: 0.1604 - g_loss: 4.6610 -
D(x|y): 0.5010 - D(G(z|y)): 0.0899 - KL Divergence: 4.6990
```

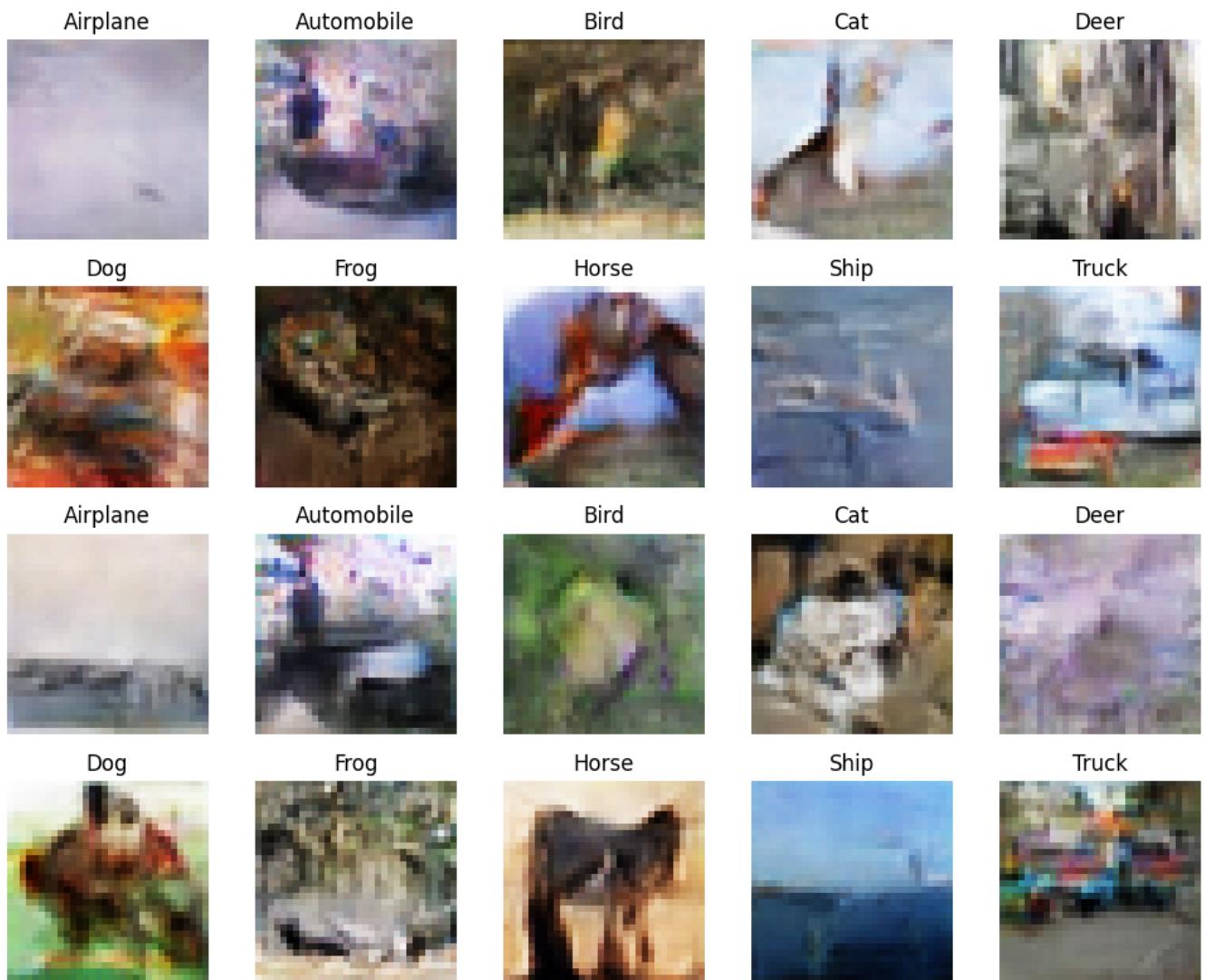
Epoch 54/200

```
782/782 [=====] - 87s 111ms/step - d_loss: 0.1650 - g_loss: 4.7243 -
D(x|y): 0.5000 - D(G(z|y)): 0.0879 - KL Divergence: 4.8778
```

Epoch 55/200

```
782/782 [=====] - 83s 106ms/step - d_loss: 0.1607 - g_loss: 4.7335 -
D(x|y): 0.4996 - D(G(z|y)): 0.0901 - KL Divergence: 4.7149
```

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-55.h5

Epoch 56/200

782/782 [=====] - 87s 111ms/step - d_loss: 0.1627 - g_loss: 4.7537 - D(x|y): 0.5003 - D(G(z|y)): 0.0859 - KL Divergence: 4.6821

Epoch 57/200

782/782 [=====] - 83s 106ms/step - d_loss: 0.1559 - g_loss: 4.8275 - D(x|y): 0.5004 - D(G(z|y)): 0.0832 - KL Divergence: 4.7856

Epoch 58/200

782/782 [=====] - 83s 106ms/step - d_loss: 0.1535 - g_loss: 4.8816 - D(x|y): 0.5007 - D(G(z|y)): 0.0798 - KL Divergence: 4.7271

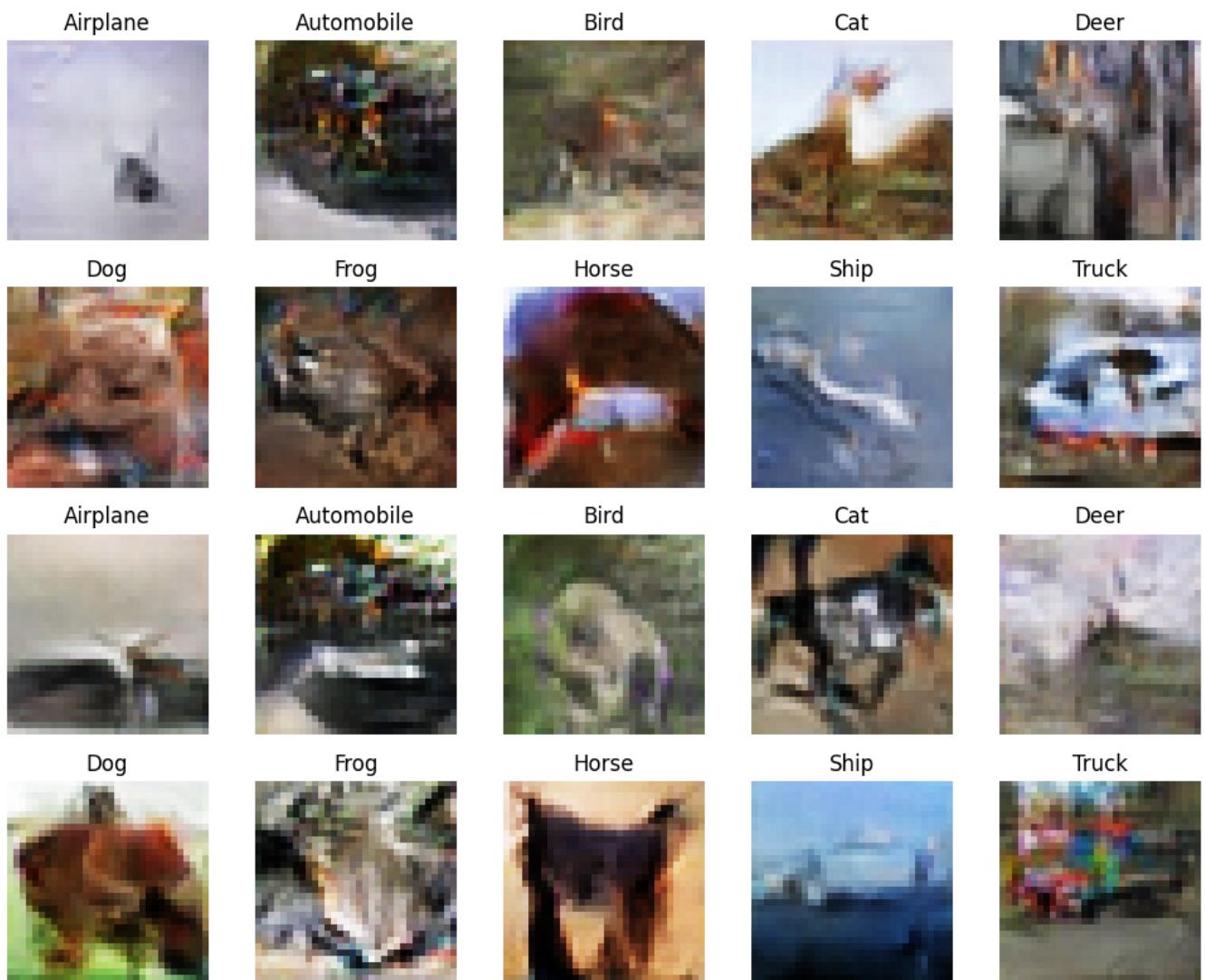
Epoch 59/200

782/782 [=====] - 84s 107ms/step - d_loss: 0.1500 - g_loss: 4.9771 - D(x|y): 0.5009 - D(G(z|y)): 0.0817 - KL Divergence: 4.8199

Epoch 60/200

782/782 [=====] - 88s 112ms/step - d_loss: 0.1500 - g_loss: 4.9952 - D(x|y): 0.4999 - D(G(z|y)): 0.0815 - KL Divergence: 4.6546

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New ACGAN/generator-epoch-60.h5

Epoch 61/200

782/782 [=====] - 86s 109ms/step - d_loss: 0.1462 - g_loss: 5.0610 - D(x|y): 0.5003 - D(G(z|y)): 0.0808 - KL Divergence: 4.8841

Epoch 62/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1453 - g_loss: 5.2278 - D(x|y): 0.5004 - D(G(z|y)): 0.0763 - KL Divergence: 4.5913

Epoch 63/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1516 - g_loss: 5.1452 - D(x|y): 0.5004 - D(G(z|y)): 0.0792 - KL Divergence: 4.6823

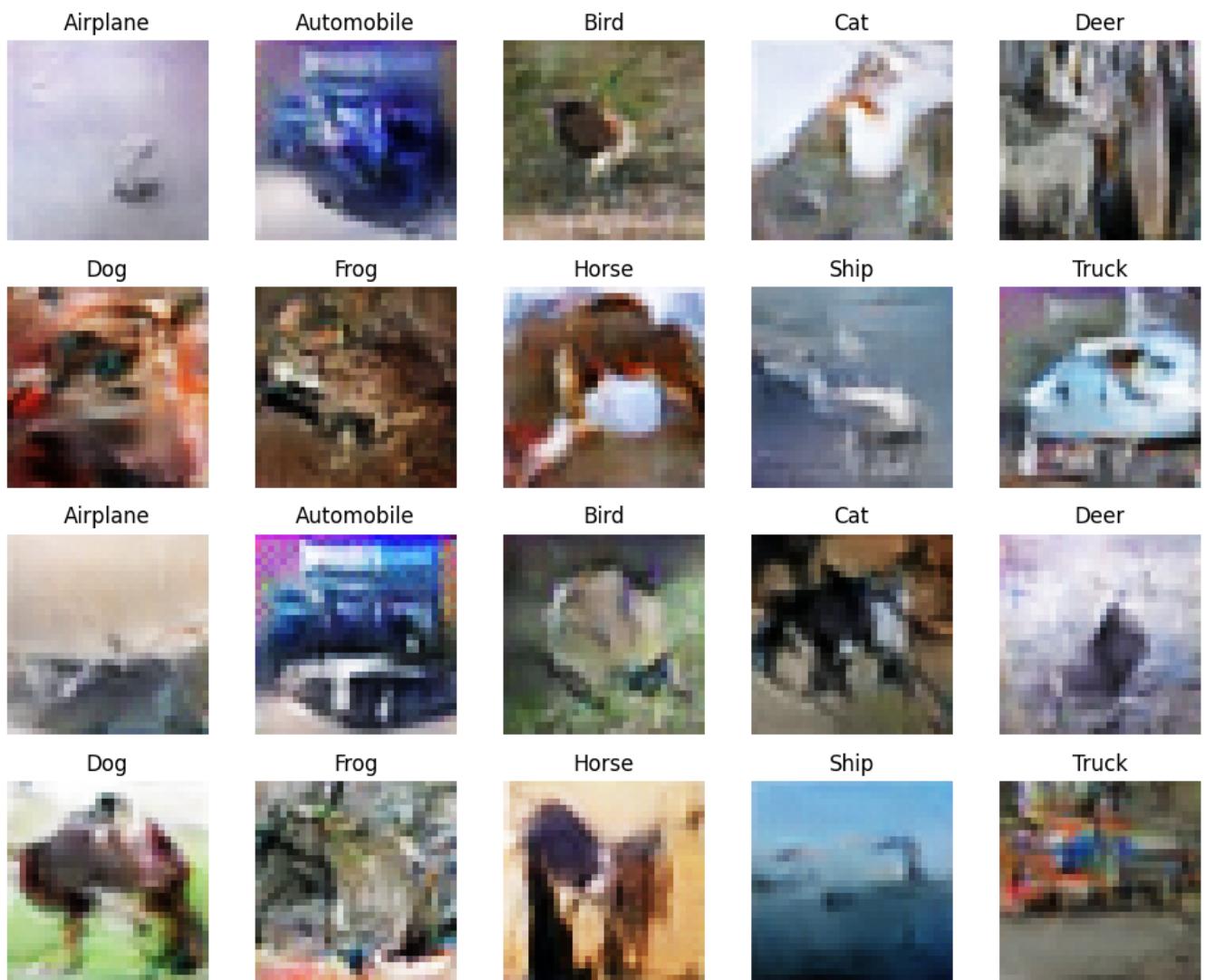
Epoch 64/200

782/782 [=====] - 89s 114ms/step - d_loss: 0.1469 - g_loss: 5.1944 - D(x|y): 0.5003 - D(G(z|y)): 0.0780 - KL Divergence: 4.6694

Epoch 65/200

782/782 [=====] - 87s 111ms/step - d_loss: 0.1439 - g_loss: 5.2272 - D(x|y): 0.5002 - D(G(z|y)): 0.0791 - KL Divergence: 4.7963

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-65.h5

Epoch 66/200

```
782/782 [=====] - 85s 109ms/step - d_loss: 0.1508 - g_loss: 5.2257 -
D(x|y): 0.5000 - D(G(z|y)): 0.0777 - KL Divergence: 4.6924
```

Epoch 67/200

```
782/782 [=====] - 85s 109ms/step - d_loss: 0.1391 - g_loss: 5.3429 -
D(x|y): 0.5004 - D(G(z|y)): 0.0753 - KL Divergence: 4.4160
```

Epoch 68/200

```
782/782 [=====] - 84s 108ms/step - d_loss: 0.1472 - g_loss: 5.3421 -
D(x|y): 0.5001 - D(G(z|y)): 0.0777 - KL Divergence: 4.8705
```

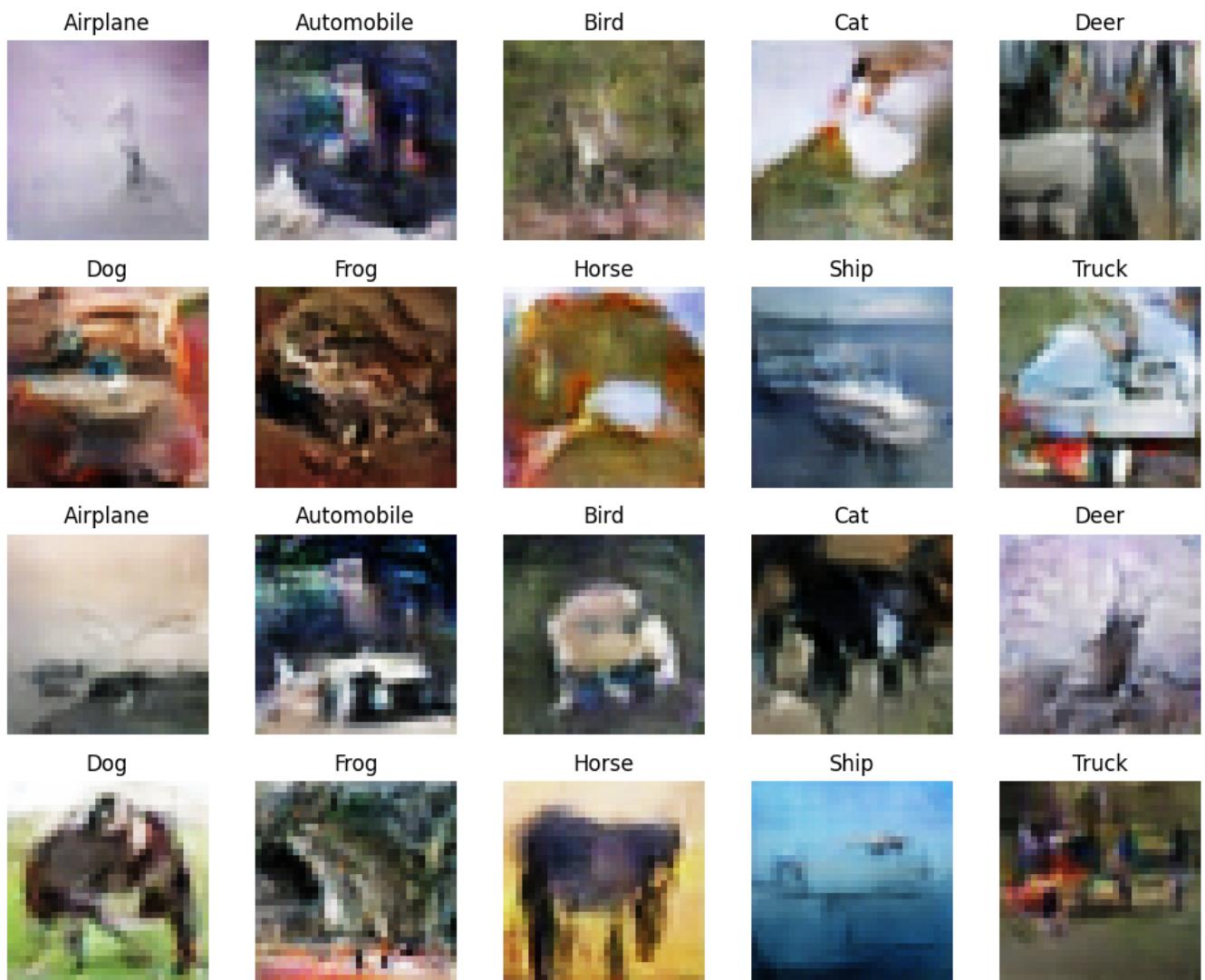
Epoch 69/200

```
782/782 [=====] - 85s 109ms/step - d_loss: 0.1397 - g_loss: 5.4528 -
D(x|y): 0.4999 - D(G(z|y)): 0.0721 - KL Divergence: 4.6233
```

Epoch 70/200

```
782/782 [=====] - 85s 108ms/step - d_loss: 0.1276 - g_loss: 5.5411 -
D(x|y): 0.5002 - D(G(z|y)): 0.0698 - KL Divergence: 4.6890
```

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New ACGAN/generator-epoch-70.h5

Epoch 71/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1285 - g_loss: 5.4968 - D(x|y): 0.5003 - D(G(z|y)): 0.0708 - KL Divergence: 4.8176

Epoch 72/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1291 - g_loss: 5.5212 - D(x|y): 0.5001 - D(G(z|y)): 0.0713 - KL Divergence: 4.4357

Epoch 73/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1294 - g_loss: 5.7067 - D(x|y): 0.5003 - D(G(z|y)): 0.0694 - KL Divergence: 4.6374

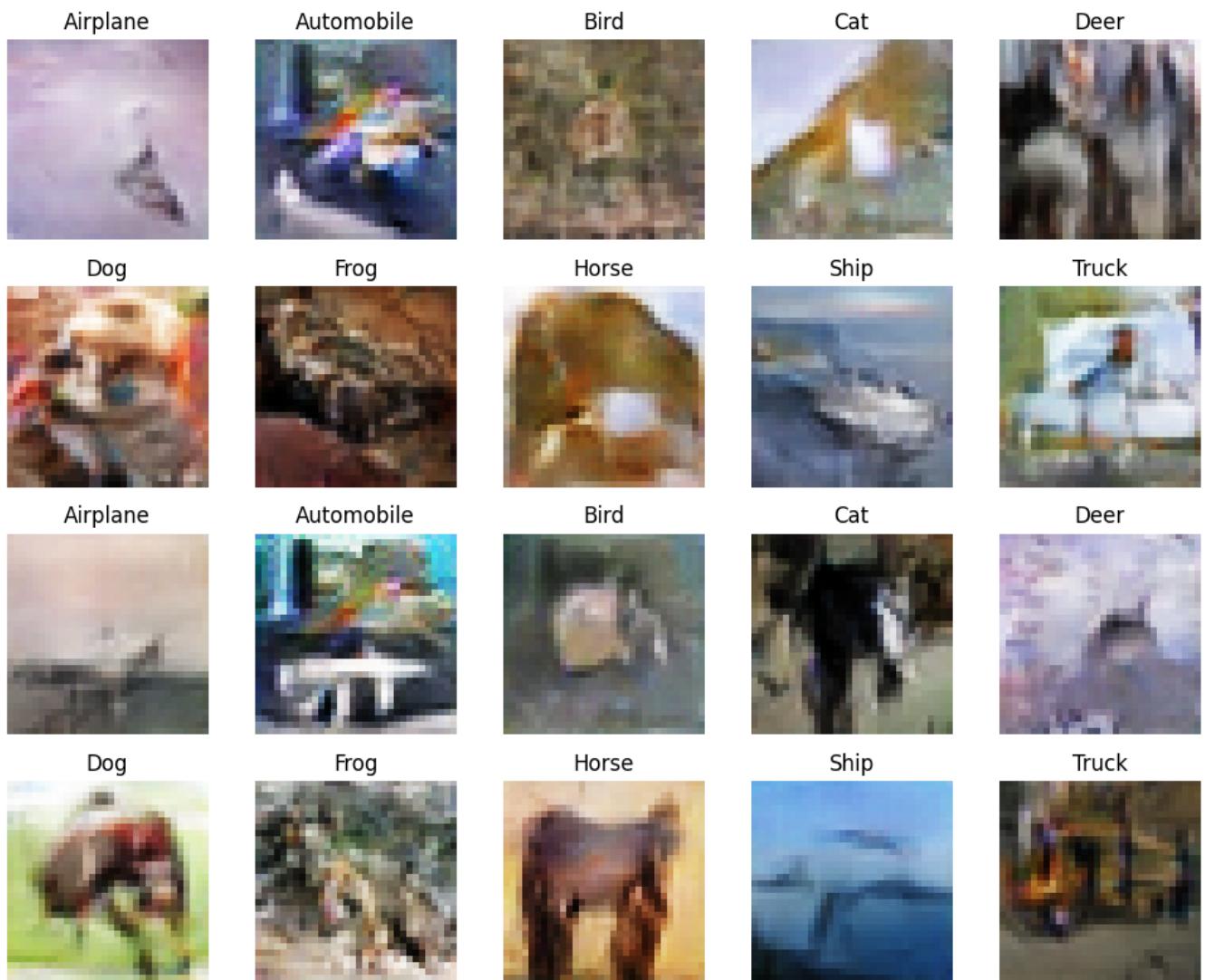
Epoch 74/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1359 - g_loss: 5.6191 - D(x|y): 0.5007 - D(G(z|y)): 0.0708 - KL Divergence: 4.4713

Epoch 75/200

782/782 [=====] - 84s 108ms/step - d_loss: 0.1278 - g_loss: 5.7401 - D(x|y): 0.5002 - D(G(z|y)): 0.0716 - KL Divergence: 4.7517

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New ACGAN/generator-epoch-75.h5

Epoch 76/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1378 - g_loss: 5.7418 - D(x|y): 0.5003 - D(G(z|y)): 0.0707 - KL Divergence: 4.6274

Epoch 77/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1273 - g_loss: 5.7199 - D(x|y): 0.5001 - D(G(z|y)): 0.0693 - KL Divergence: 4.6511

Epoch 78/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1281 - g_loss: 5.8291 - D(x|y): 0.4996 - D(G(z|y)): 0.0665 - KL Divergence: 4.5333

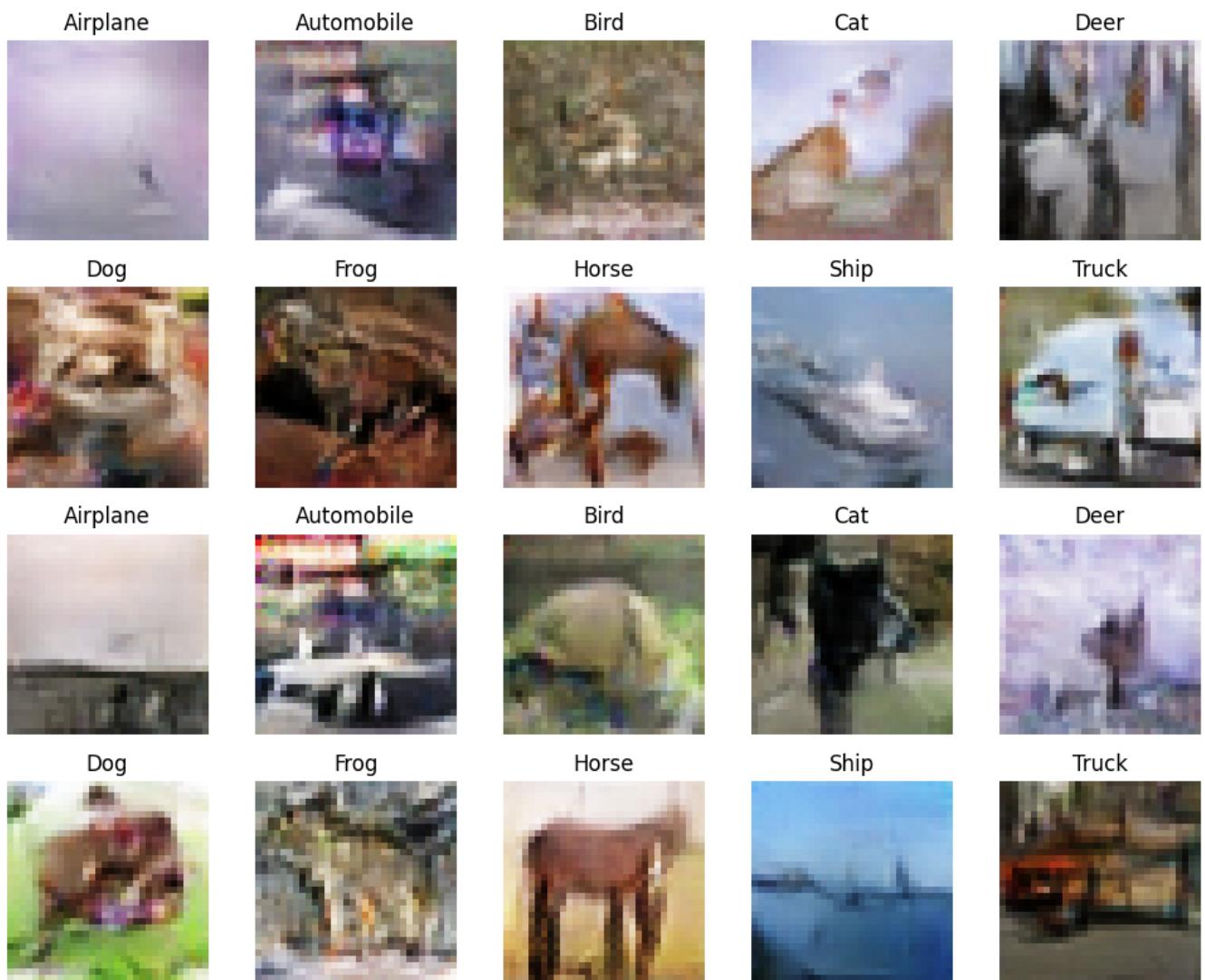
Epoch 79/200

782/782 [=====] - 84s 108ms/step - d_loss: 0.1251 - g_loss: 5.8501 - D(x|y): 0.5004 - D(G(z|y)): 0.0646 - KL Divergence: 4.5812

Epoch 80/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1225 - g_loss: 5.9108 - D(x|y): 0.4998 - D(G(z|y)): 0.0642 - KL Divergence: 4.7039

1/1 [=====] - 0s 21ms/step



Generator Checkpoint - New ACGAN/generator-epoch-80.h5

Epoch 81/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1214 - g_loss: 5.9751 - D(x|y): 0.5003 - D(G(z|y)): 0.0649 - KL Divergence: 4.5269

Epoch 82/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.1209 - g_loss: 5.9426 - D(x|y): 0.5001 - D(G(z|y)): 0.0664 - KL Divergence: 4.7336

Epoch 83/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1170 - g_loss: 6.0338 - D(x|y): 0.5002 - D(G(z|y)): 0.0630 - KL Divergence: 4.5350

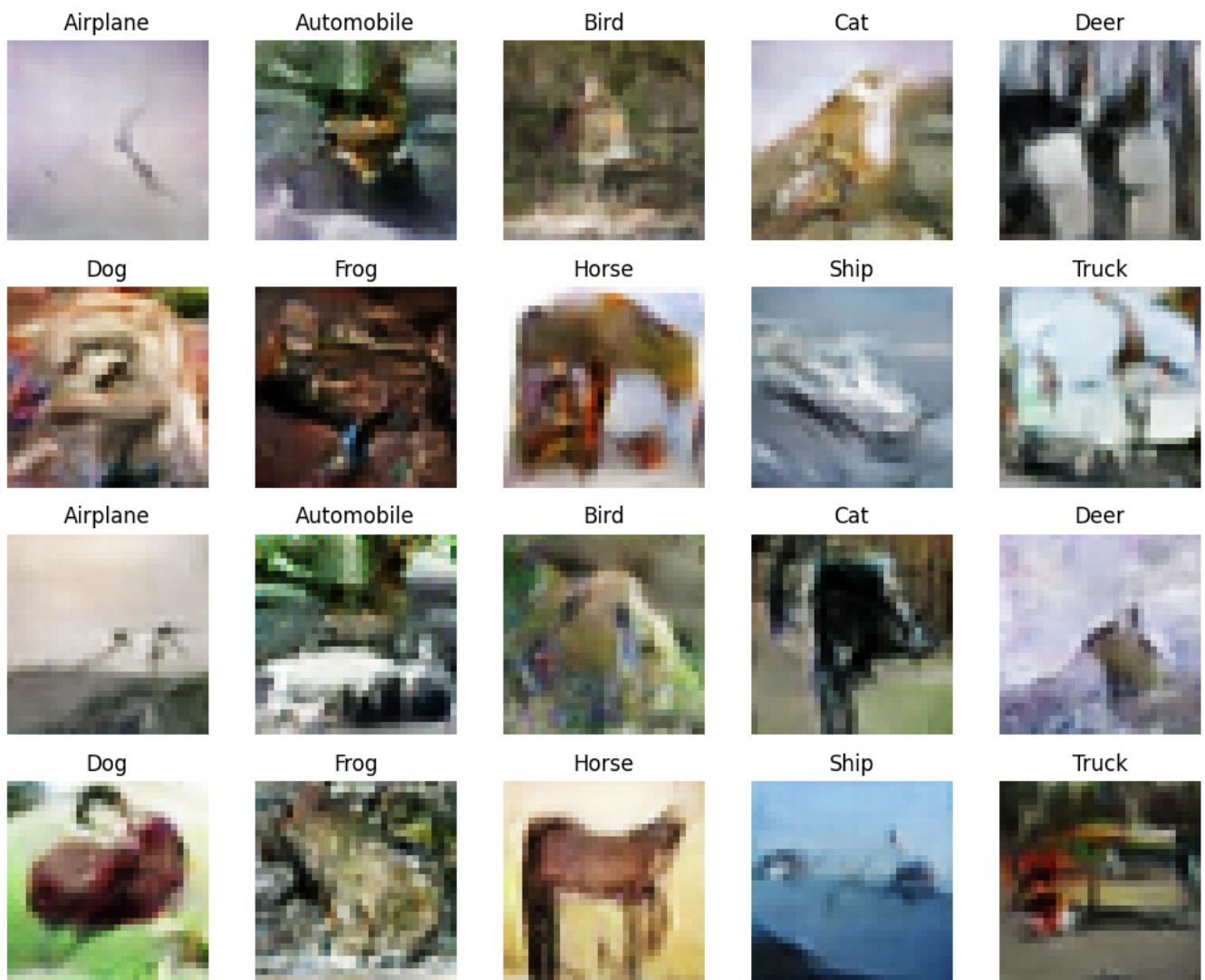
Epoch 84/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1284 - g_loss: 6.0368 - D(x|y): 0.5007 - D(G(z|y)): 0.0635 - KL Divergence: 4.7853

Epoch 85/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1166 - g_loss: 6.0800 - D(x|y): 0.5006 - D(G(z|y)): 0.0616 - KL Divergence: 4.5917

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-85.h5

Epoch 86/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1221 - g_loss: 6.1005 - D(x|y): 0.5000 - D(G(z|y)): 0.0621 - KL Divergence: 4.7245

Epoch 87/200

782/782 [=====] - 84s 108ms/step - d_loss: 0.1205 - g_loss: 6.1454 - D(x|y): 0.5004 - D(G(z|y)): 0.0610 - KL Divergence: 4.6772

Epoch 88/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1162 - g_loss: 6.1645 - D(x|y): 0.5000 - D(G(z|y)): 0.0619 - KL Divergence: 4.5167

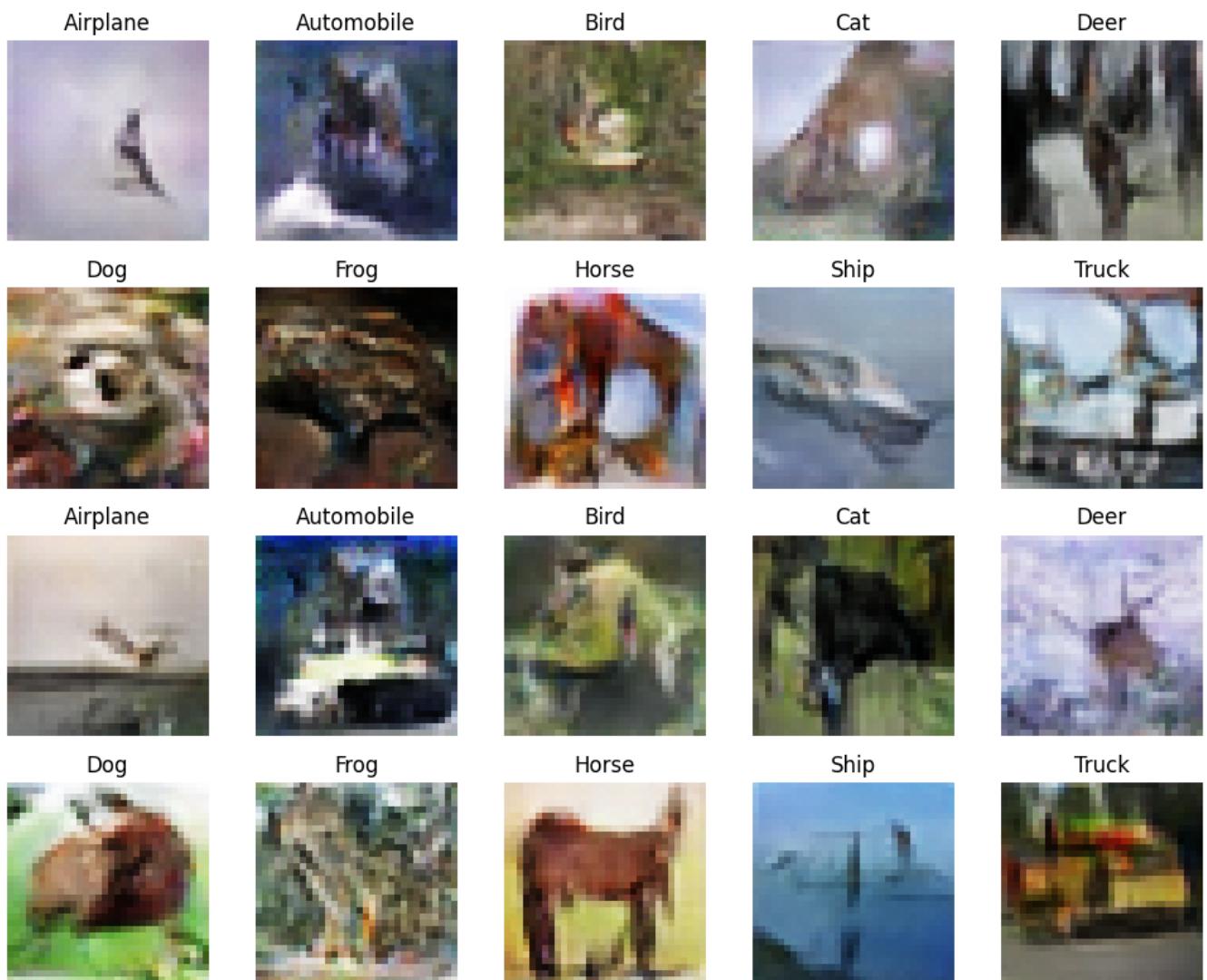
Epoch 89/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1151 - g_loss: 6.2019 - D(x|y): 0.5002 - D(G(z|y)): 0.0594 - KL Divergence: 4.4771

Epoch 90/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1173 - g_loss: 6.2635 - D(x|y): 0.5006 - D(G(z|y)): 0.0594 - KL Divergence: 4.6129

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-90.h5

Epoch 91/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1204 - g_loss: 6.1954 - D(x|y): 0.5006 - D(G(z|y)): 0.0617 - KL Divergence: 4.5797

Epoch 92/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1158 - g_loss: 6.2601 - D(x|y): 0.5003 - D(G(z|y)): 0.0572 - KL Divergence: 4.5295

Epoch 93/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1213 - g_loss: 6.2188 - D(x|y): 0.4994 - D(G(z|y)): 0.0598 - KL Divergence: 4.7914

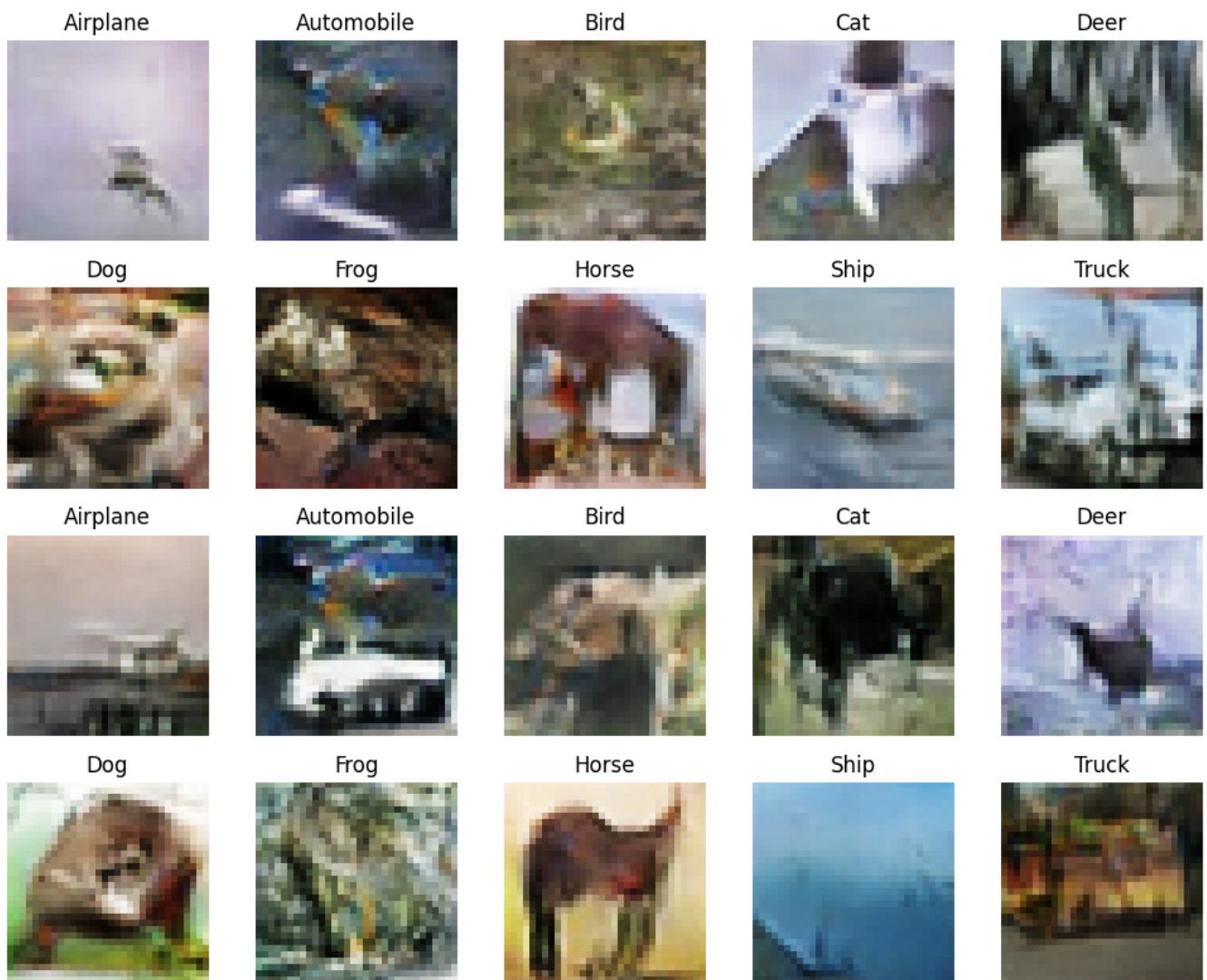
Epoch 94/200

782/782 [=====] - 84s 108ms/step - d_loss: 0.1142 - g_loss: 6.2956 - D(x|y): 0.5006 - D(G(z|y)): 0.0604 - KL Divergence: 4.4055

Epoch 95/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1134 - g_loss: 6.3181 - D(x|y): 0.5004 - D(G(z|y)): 0.0596 - KL Divergence: 4.7710

1/1 [=====] - 0s 22ms/step



Generator Checkpoint - New ACGAN/generator-epoch-95.h5

Epoch 96/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1156 - g_loss: 6.3443 - D(x|y): 0.5002 - D(G(z|y)): 0.0573 - KL Divergence: 4.7777

Epoch 97/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1079 - g_loss: 6.4356 - D(x|y): 0.5005 - D(G(z|y)): 0.0582 - KL Divergence: 4.4516

Epoch 98/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1055 - g_loss: 6.4733 - D(x|y): 0.5007 - D(G(z|y)): 0.0562 - KL Divergence: 4.6866

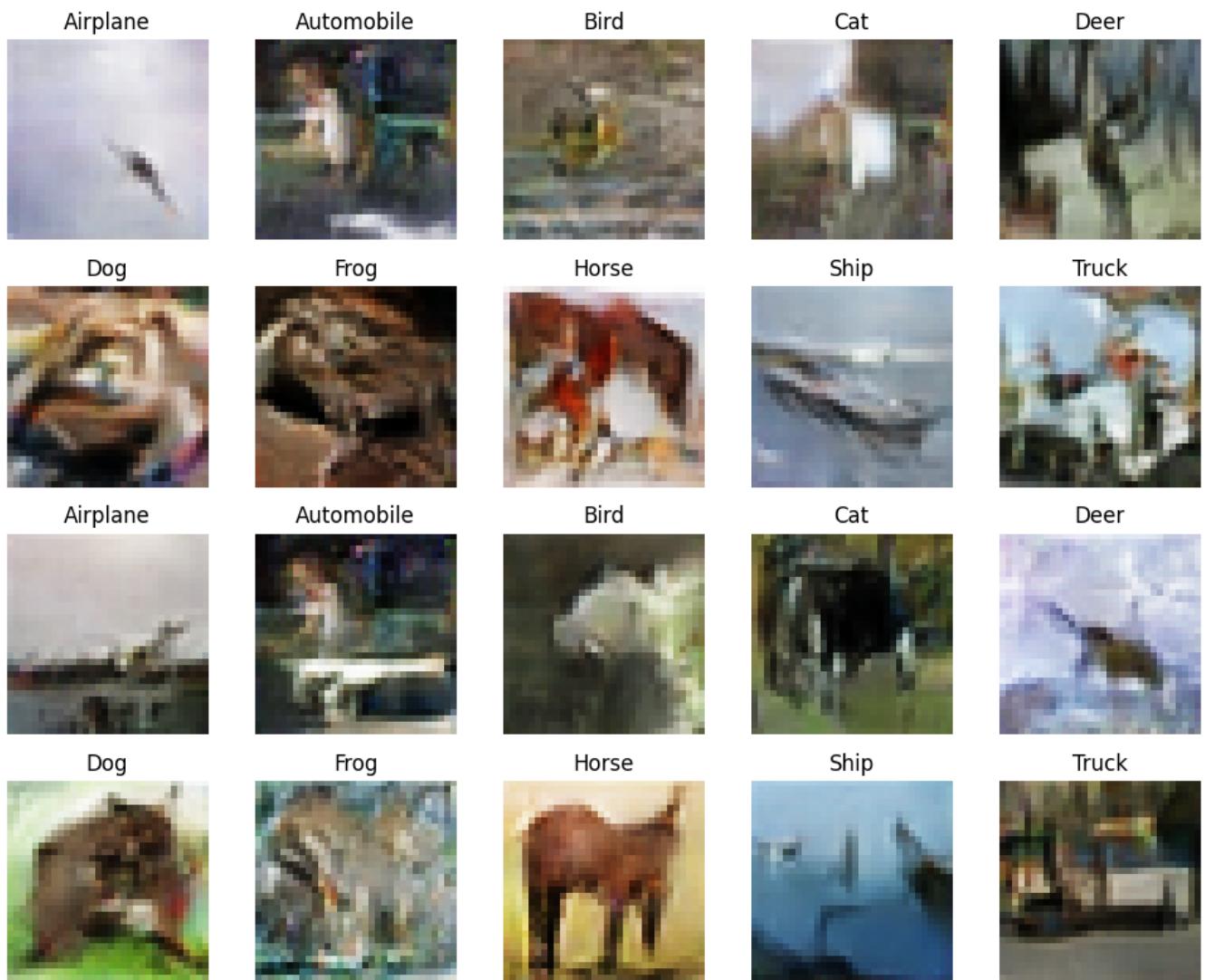
Epoch 99/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1186 - g_loss: 6.4040 - D(x|y): 0.5006 - D(G(z|y)): 0.0592 - KL Divergence: 4.7414

Epoch 100/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1084 - g_loss: 6.5223 - D(x|y): 0.5000 - D(G(z|y)): 0.0544 - KL Divergence: 4.6303

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - New ACGAN/generator-epoch-100.h5

Epoch 101/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1034 - g_loss: 6.6513 - D(x|y): 0.5006 - D(G(z|y)): 0.0527 - KL Divergence: 4.7078

Epoch 102/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1115 - g_loss: 6.5589 - D(x|y): 0.5005 - D(G(z|y)): 0.0587 - KL Divergence: 4.5346

Epoch 103/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1037 - g_loss: 6.6209 - D(x|y): 0.5002 - D(G(z|y)): 0.0560 - KL Divergence: 4.6566

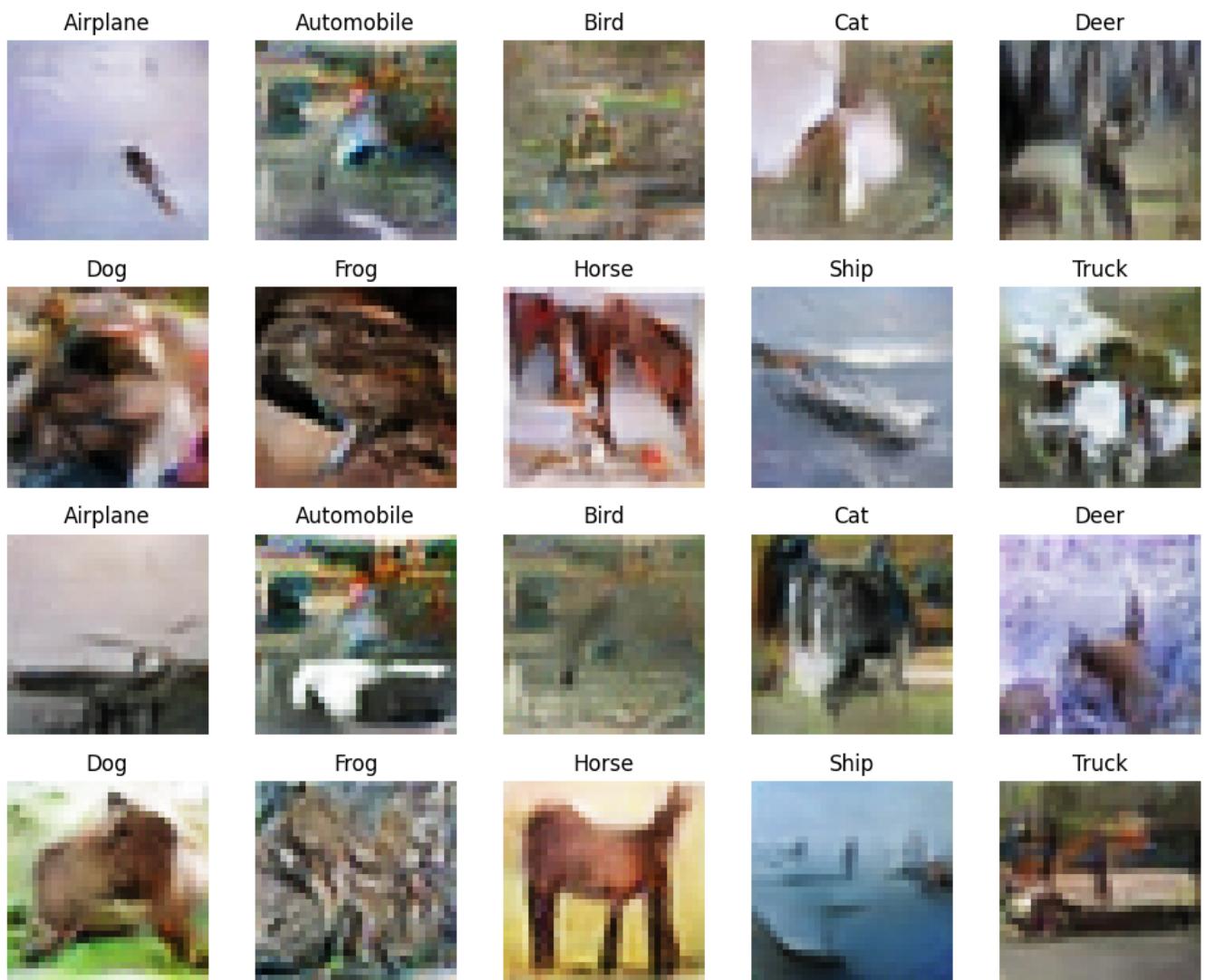
Epoch 104/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.1182 - g_loss: 6.6285 - D(x|y): 0.5004 - D(G(z|y)): 0.0561 - KL Divergence: 4.5008

Epoch 105/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1034 - g_loss: 6.7122 - D(x|y): 0.5003 - D(G(z|y)): 0.0531 - KL Divergence: 4.7357

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - New ACGAN/generator-epoch-105.h5

Epoch 106/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.1060 - g_loss: 6.6175 - D(x|y): 0.4998 - D(G(z|y)): 0.0555 - KL Divergence: 4.4286

Epoch 107/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0973 - g_loss: 6.7946 - D(x|y): 0.5000 - D(G(z|y)): 0.0526 - KL Divergence: 4.4410

Epoch 108/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1024 - g_loss: 6.7748 - D(x|y): 0.5001 - D(G(z|y)): 0.0525 - KL Divergence: 4.5522

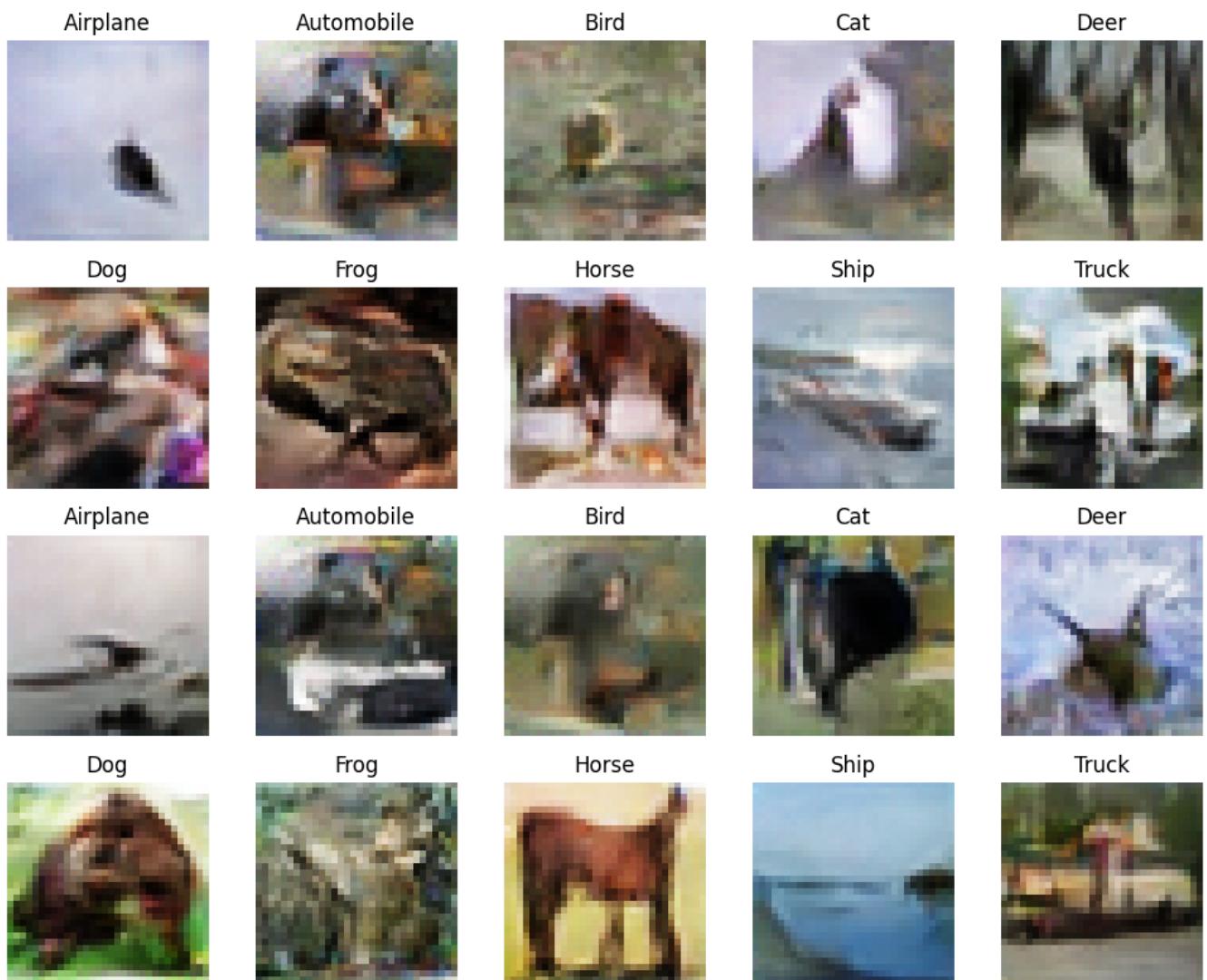
Epoch 109/200

782/782 [=====] - 92s 117ms/step - d_loss: 0.1066 - g_loss: 6.8691 - D(x|y): 0.5003 - D(G(z|y)): 0.0522 - KL Divergence: 4.7278

Epoch 110/200

782/782 [=====] - 84s 108ms/step - d_loss: 0.1013 - g_loss: 6.8318 - D(x|y): 0.4997 - D(G(z|y)): 0.0513 - KL Divergence: 4.4754

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New ACGAN/generator-epoch-110.h5

Epoch 111/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1022 - g_loss: 6.8294 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0527 - KL Divergence: 4.6183

Epoch 112/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1055 - g_loss: 6.9106 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0485 - KL Divergence: 4.6868

Epoch 113/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1006 - g_loss: 6.9977 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0496 - KL Divergence: 4.4599

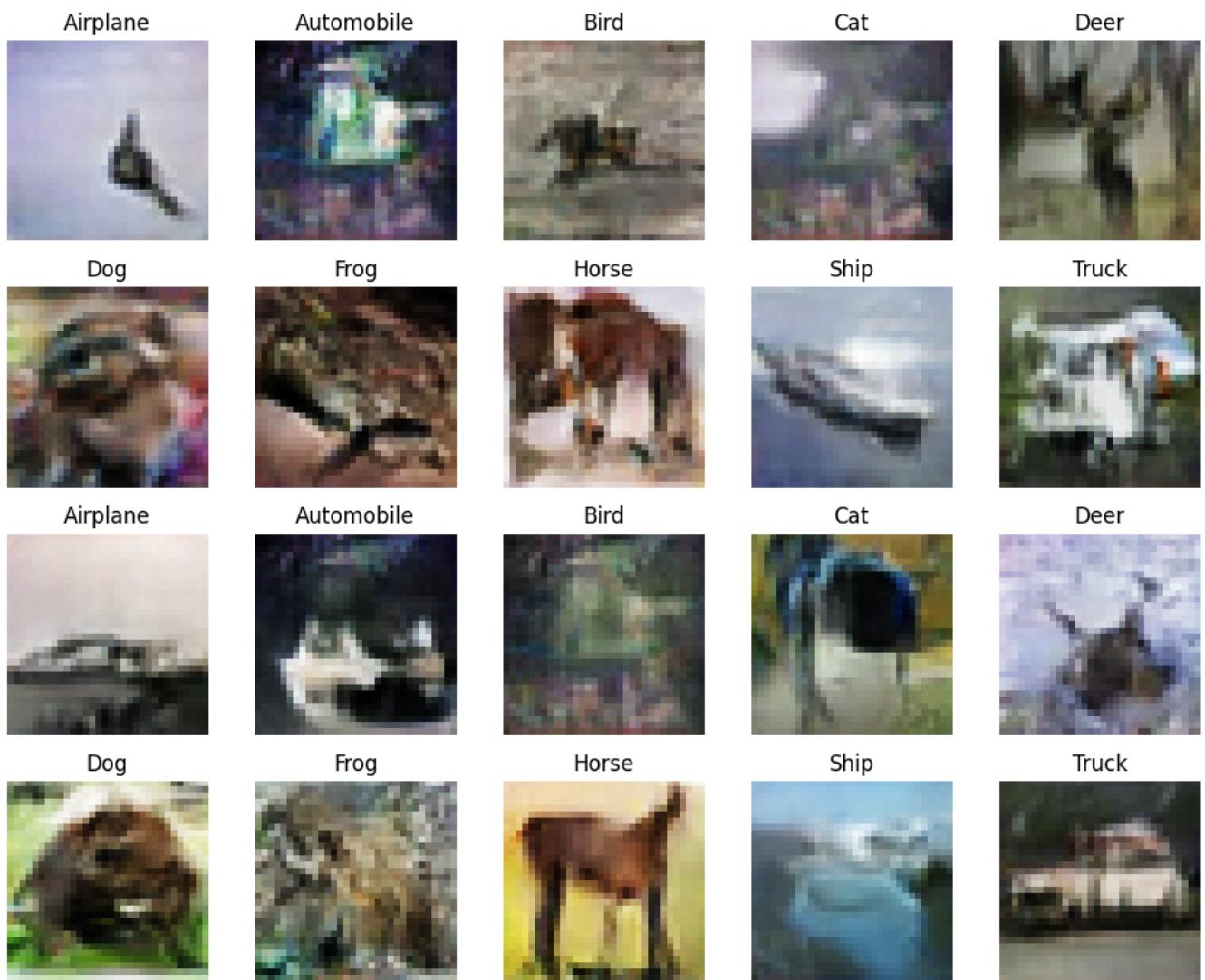
Epoch 114/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1004 - g_loss: 6.9846 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0511 - KL Divergence: 4.7142

Epoch 115/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.1025 - g_loss: 6.8295 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0539 - KL Divergence: 4.9154

1/1 [=====] - 0s 33ms/step



Generator Checkpoint - New ACGAN/generator-epoch-115.h5

Epoch 116/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0939 - g_loss: 7.0155 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0501$ - KL Divergence: 4.8058

Epoch 117/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0981 - g_loss: 7.0627 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.0471$ - KL Divergence: 4.3865

Epoch 118/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0944 - g_loss: 7.1288 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0474$ - KL Divergence: 4.5317

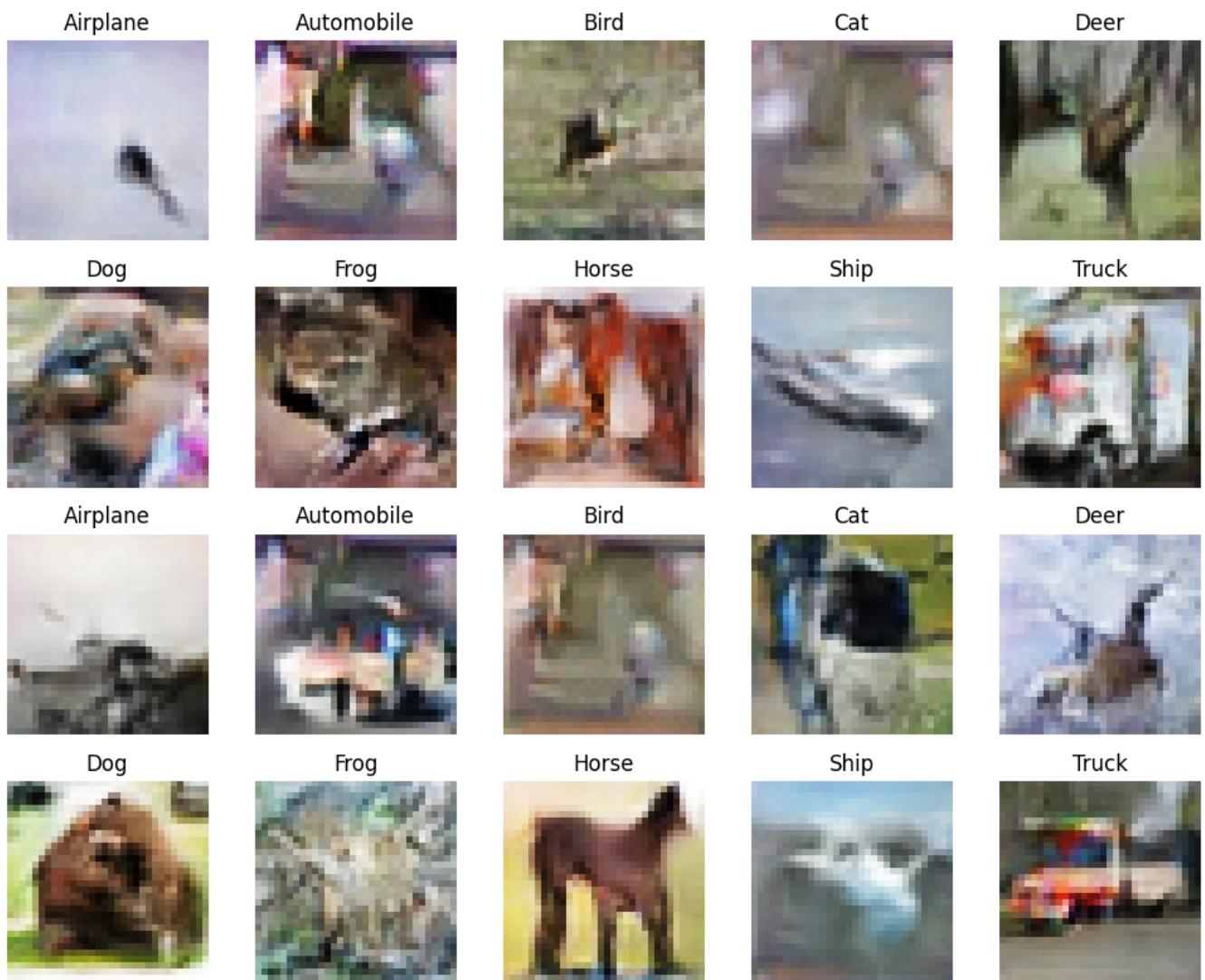
Epoch 119/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0917 - g_loss: 6.9937 -
 $D(x|y): 0.5007$ - $D(G(z|y)): 0.0482$ - KL Divergence: 4.5877

Epoch 120/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0945 - g_loss: 7.0966 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.0465$ - KL Divergence: 4.7501

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - New ACGAN/generator-epoch-120.h5

Epoch 121/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0921 - g_loss: 7.1253 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0484 - KL Divergence: 4.5716

Epoch 122/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0987 - g_loss: 7.0774 -
 $D(x|y)$: 0.5007 - $D(G(z|y))$: 0.0490 - KL Divergence: 4.6552

Epoch 123/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.0926 - g_loss: 7.1581 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0472 - KL Divergence: 4.4874

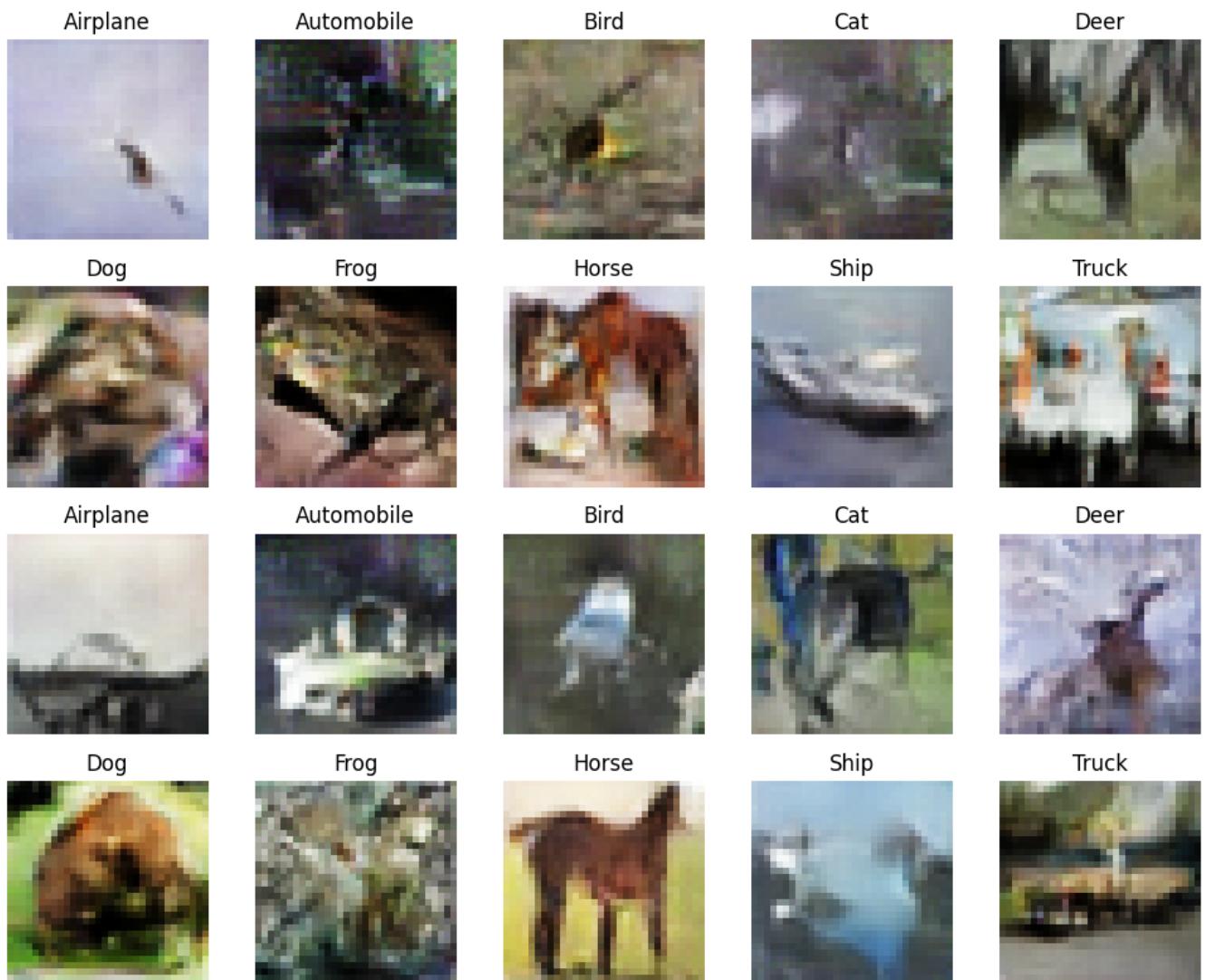
Epoch 124/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0906 - g_loss: 7.1482 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0474 - KL Divergence: 4.7087

Epoch 125/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0925 - g_loss: 7.1977 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0470 - KL Divergence: 4.6610

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-125.h5

Epoch 126/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.0942 - g_loss: 7.0984 - D(x|y): 0.5004 - D(G(z|y)): 0.0501 - KL Divergence: 4.6217

Epoch 127/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.0919 - g_loss: 7.2160 - D(x|y): 0.5000 - D(G(z|y)): 0.0458 - KL Divergence: 4.2903

Epoch 128/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0896 - g_loss: 7.3746 - D(x|y): 0.5003 - D(G(z|y)): 0.0443 - KL Divergence: 4.6586

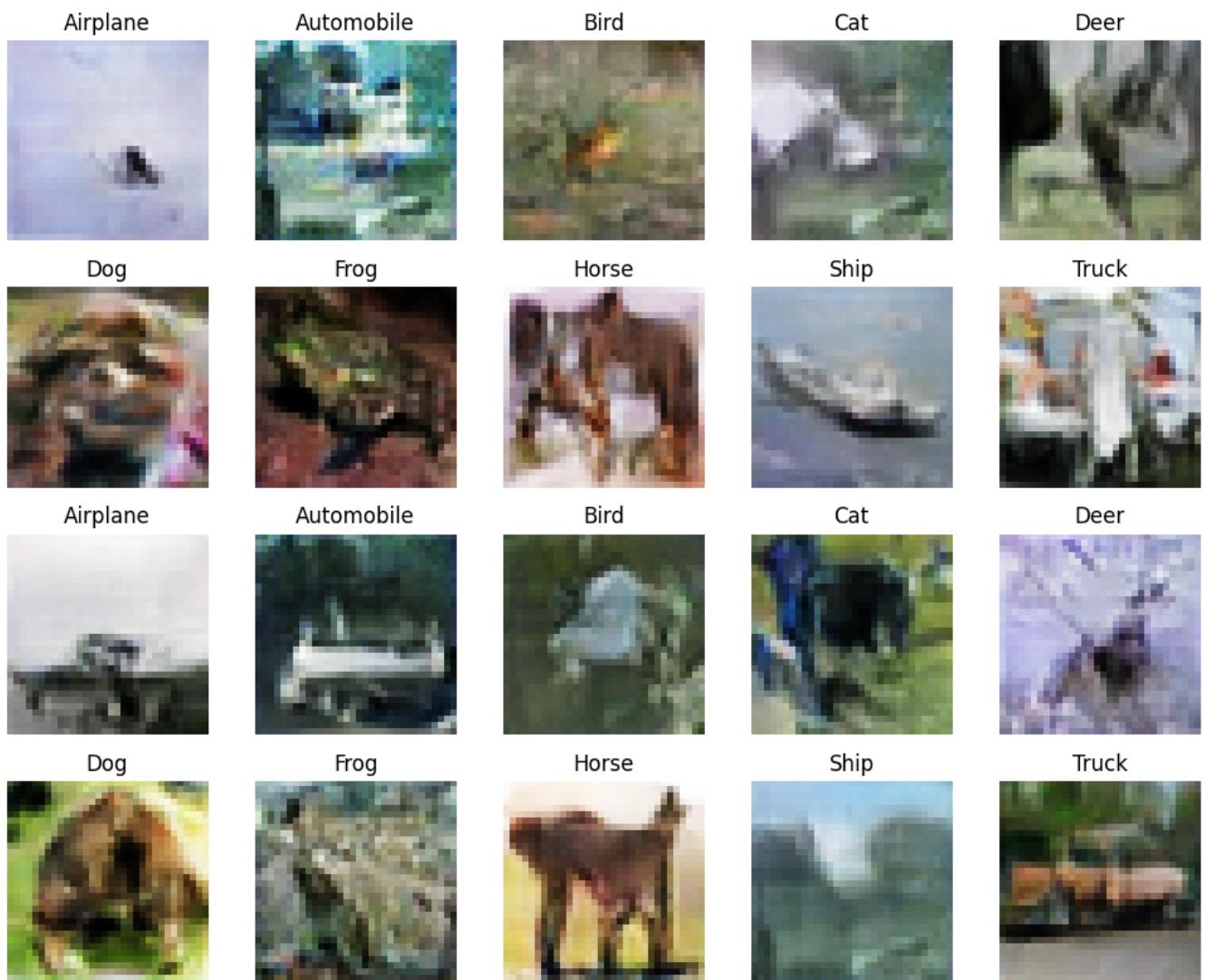
Epoch 129/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0870 - g_loss: 7.3901 - D(x|y): 0.5005 - D(G(z|y)): 0.0455 - KL Divergence: 4.5553

Epoch 130/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0919 - g_loss: 7.4774 - D(x|y): 0.5000 - D(G(z|y)): 0.0452 - KL Divergence: 4.5623

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New ACGAN/generator-epoch-130.h5

Epoch 131/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0871 - g_loss: 7.3757 - D(x|y): 0.5007 - D(G(z|y)): 0.0462 - KL Divergence: 4.5088

Epoch 132/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0879 - g_loss: 7.4005 - D(x|y): 0.4999 - D(G(z|y)): 0.0433 - KL Divergence: 4.5512

Epoch 133/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0818 - g_loss: 7.4716 - D(x|y): 0.5003 - D(G(z|y)): 0.0412 - KL Divergence: 4.3727

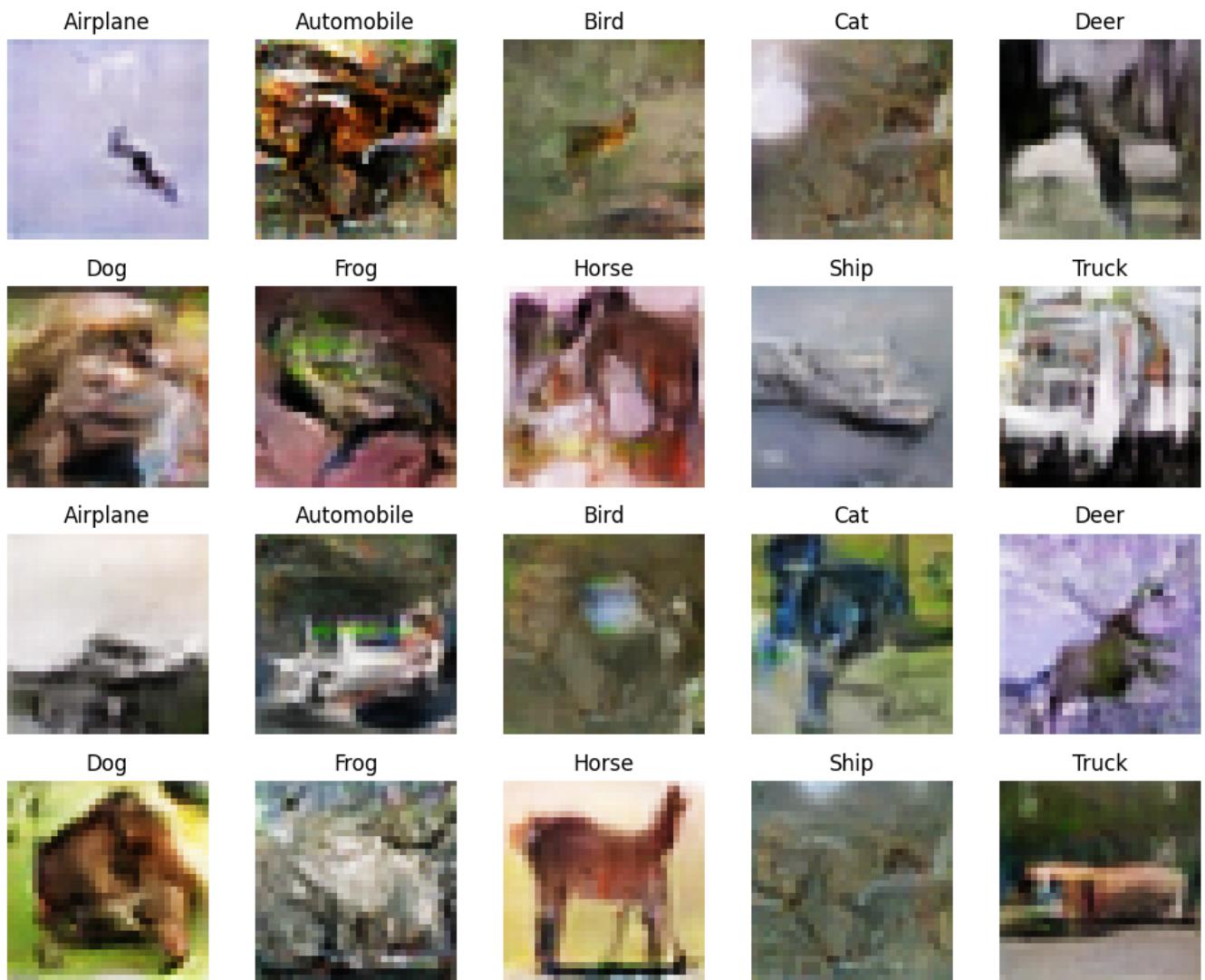
Epoch 134/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0855 - g_loss: 7.5332 - D(x|y): 0.5003 - D(G(z|y)): 0.0446 - KL Divergence: 4.7328

Epoch 135/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0894 - g_loss: 7.3797 - D(x|y): 0.5000 - D(G(z|y)): 0.0484 - KL Divergence: 4.6957

1/1 [=====] - 0s 22ms/step



Generator Checkpoint - New ACGAN/generator-epoch-135.h5

Epoch 136/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0920 - g_loss: 7.5068 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0424 - KL Divergence: 4.8588

Epoch 137/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0914 - g_loss: 7.3360 -
 $D(x|y)$: 0.5010 - $D(G(z|y))$: 0.0461 - KL Divergence: 4.6405

Epoch 138/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0832 - g_loss: 7.5442 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0421 - KL Divergence: 4.6114

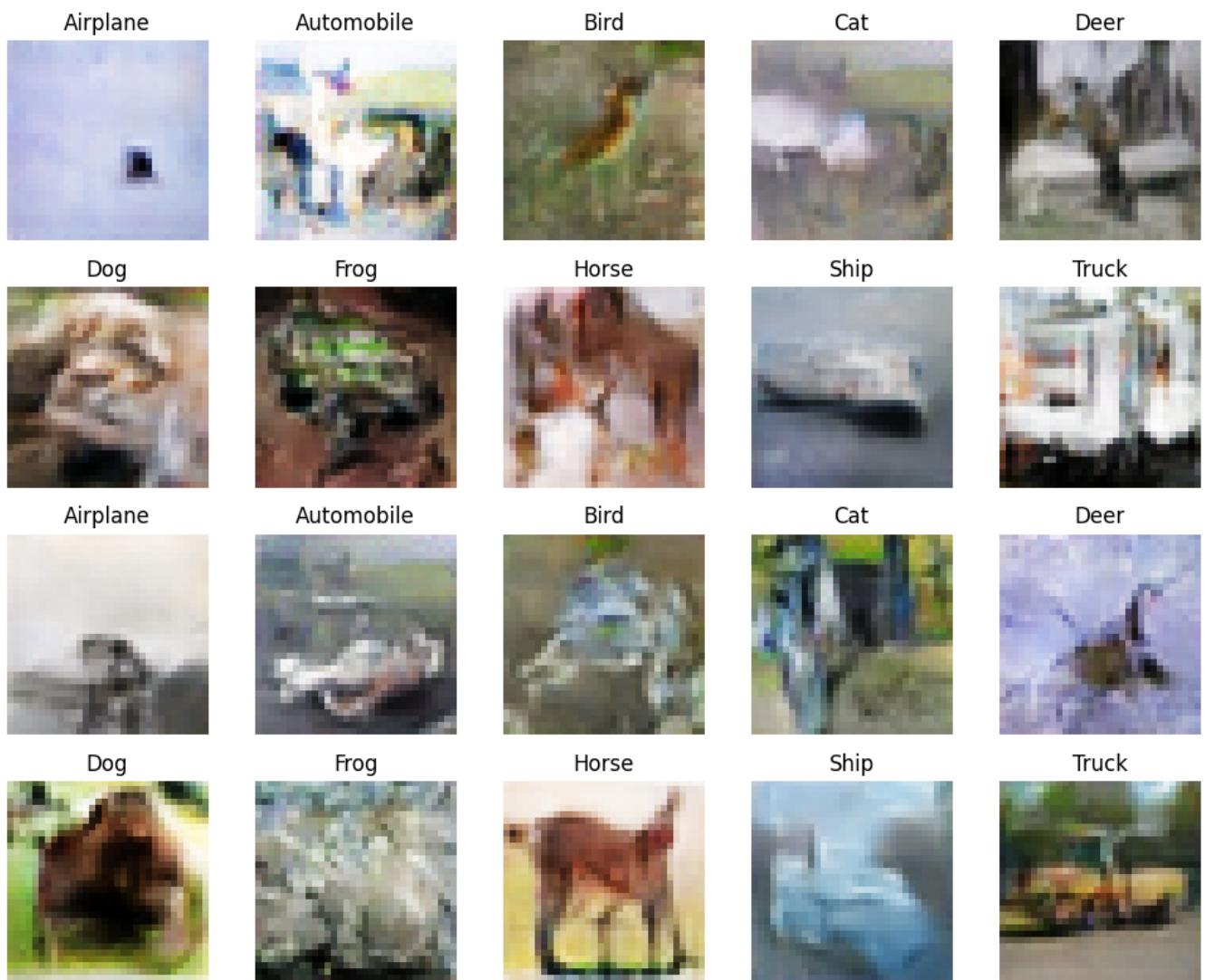
Epoch 139/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0891 - g_loss: 7.5231 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0441 - KL Divergence: 4.5388

Epoch 140/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0817 - g_loss: 7.4989 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0427 - KL Divergence: 4.5713

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-140.h5

Epoch 141/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0831 - g_loss: 7.7126 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0399 - KL Divergence: 4.4149

Epoch 142/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0883 - g_loss: 7.5766 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0424 - KL Divergence: 4.6055

Epoch 143/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0808 - g_loss: 7.5995 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0410 - KL Divergence: 4.4712

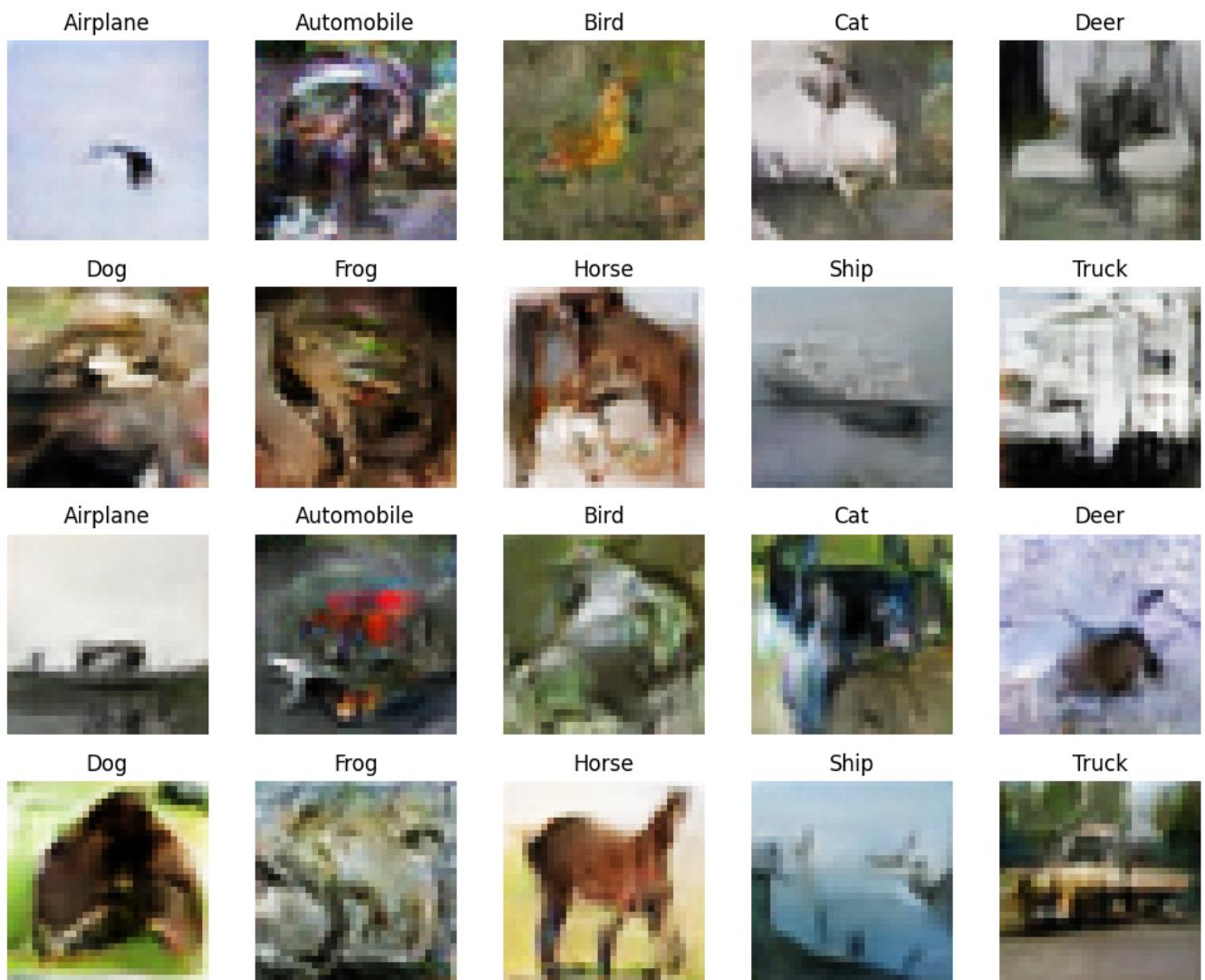
Epoch 144/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0846 - g_loss: 7.7835 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0389 - KL Divergence: 4.7681

Epoch 145/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.0846 - g_loss: 7.7035 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0398 - KL Divergence: 4.6552

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - New ACGAN/generator-epoch-145.h5

Epoch 146/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0783 - g_loss: 7.8423 - D(x|y): 0.5000 - D(G(z|y)): 0.0374 - KL Divergence: 4.5797

Epoch 147/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0869 - g_loss: 7.5578 - D(x|y): 0.5004 - D(G(z|y)): 0.0441 - KL Divergence: 4.4920

Epoch 148/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0825 - g_loss: 7.7799 - D(x|y): 0.5002 - D(G(z|y)): 0.0407 - KL Divergence: 4.5827

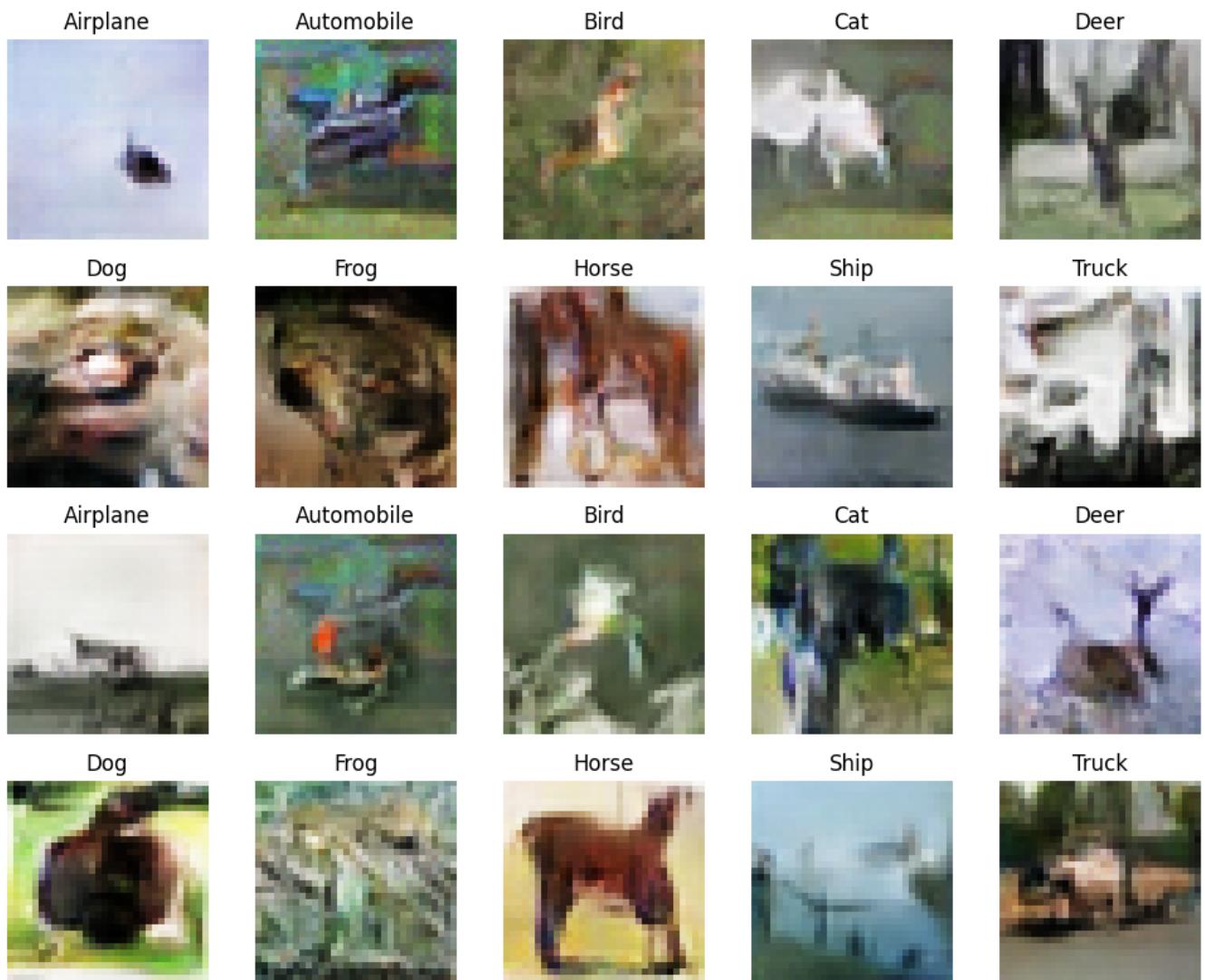
Epoch 149/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.0828 - g_loss: 7.7144 - D(x|y): 0.5002 - D(G(z|y)): 0.0416 - KL Divergence: 4.4182

Epoch 150/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0826 - g_loss: 7.7520 - D(x|y): 0.5000 - D(G(z|y)): 0.0405 - KL Divergence: 4.6064

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - New ACGAN/generator-epoch-150.h5

Epoch 151/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0835 - g_loss: 7.6843 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0419 - KL Divergence: 4.6644

Epoch 152/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0805 - g_loss: 7.7413 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0380 - KL Divergence: 4.3976

Epoch 153/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0787 - g_loss: 7.7224 -
 $D(x|y)$: 0.5007 - $D(G(z|y))$: 0.0397 - KL Divergence: 4.3669

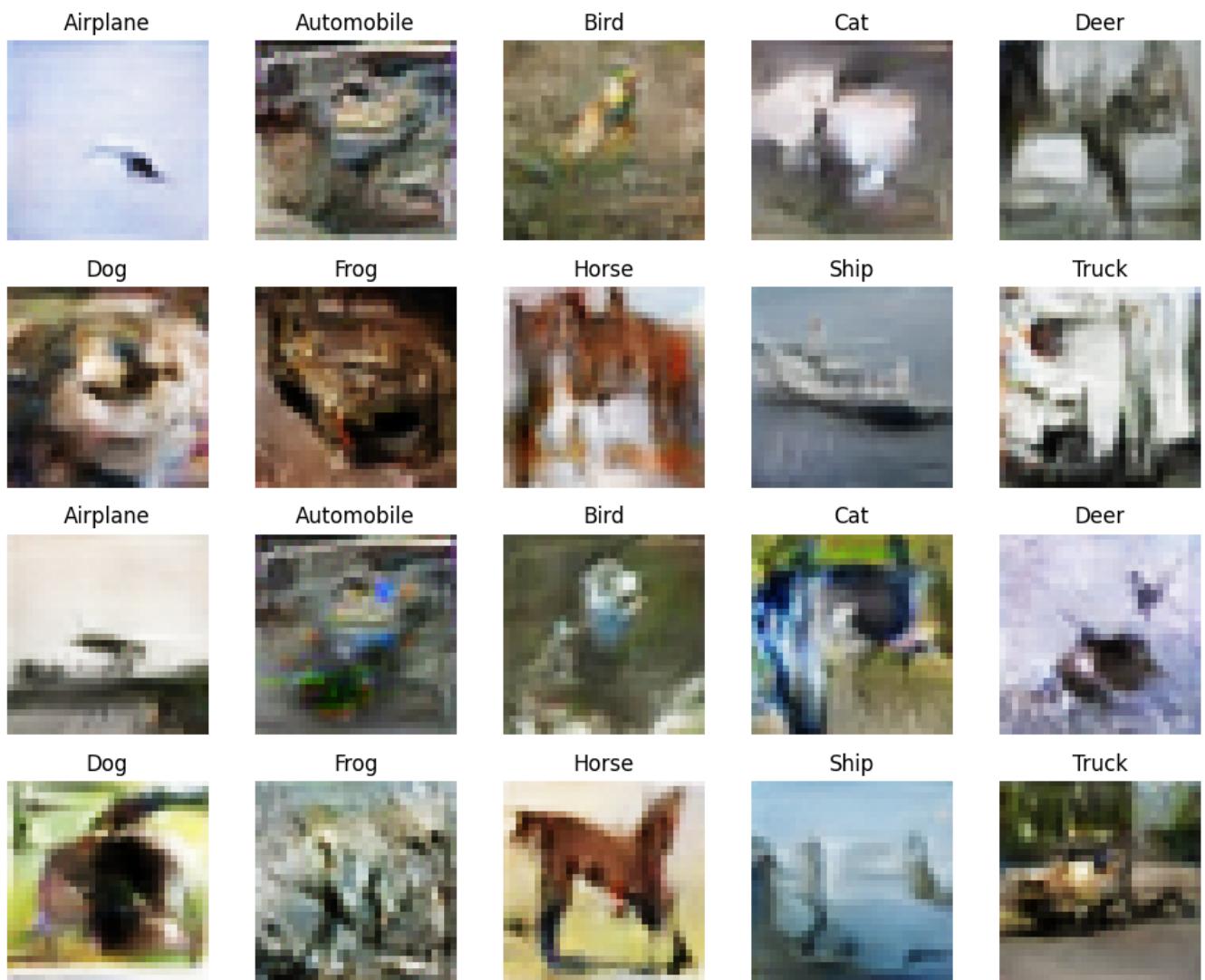
Epoch 154/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0851 - g_loss: 7.8253 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0385 - KL Divergence: 4.7479

Epoch 155/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0780 - g_loss: 7.6519 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0388 - KL Divergence: 4.8820

1/1 [=====] - 0s 22ms/step



Generator Checkpoint - New ACGAN/generator-epoch-155.h5

Epoch 156/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0798 - g_loss: 8.0198 - D(x|y): 0.5001 - D(G(z|y)): 0.0374 - KL Divergence: 4.4092

Epoch 157/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0726 - g_loss: 7.8243 - D(x|y): 0.5002 - D(G(z|y)): 0.0382 - KL Divergence: 4.7565

Epoch 158/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0805 - g_loss: 7.8146 - D(x|y): 0.5008 - D(G(z|y)): 0.0396 - KL Divergence: 4.3910

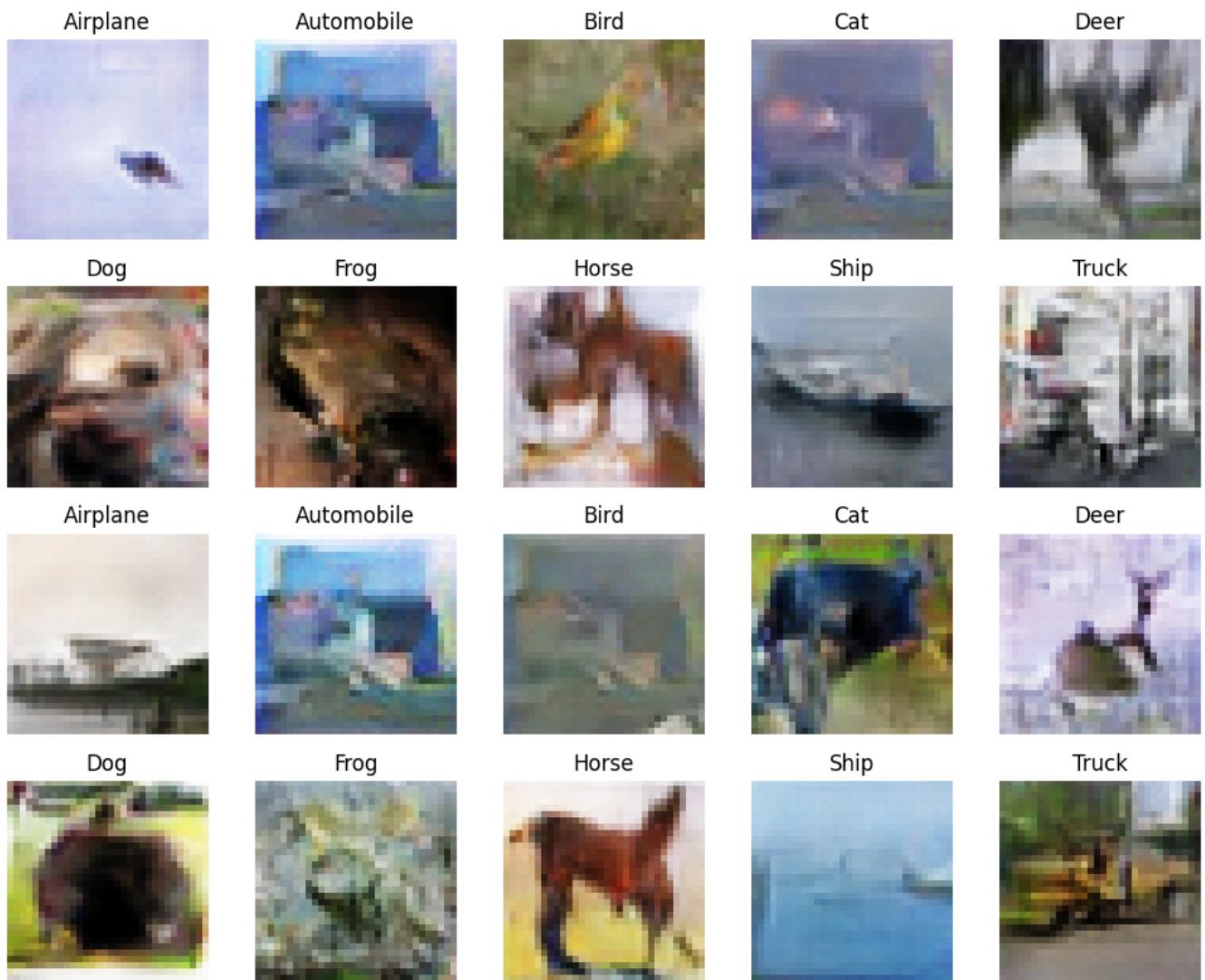
Epoch 159/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0820 - g_loss: 7.7585 - D(x|y): 0.4997 - D(G(z|y)): 0.0387 - KL Divergence: 4.5328

Epoch 160/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0779 - g_loss: 7.9329 - D(x|y): 0.5001 - D(G(z|y)): 0.0362 - KL Divergence: 4.4244

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - New ACGAN/generator-epoch-160.h5

Epoch 161/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0750 - g_loss: 7.9402 - D(x|y): 0.5004 - D(G(z|y)): 0.0364 - KL Divergence: 4.5819

Epoch 162/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0878 - g_loss: 7.7922 - D(x|y): 0.5001 - D(G(z|y)): 0.0431 - KL Divergence: 4.7076

Epoch 163/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0707 - g_loss: 8.1218 - D(x|y): 0.5003 - D(G(z|y)): 0.0345 - KL Divergence: 4.6556

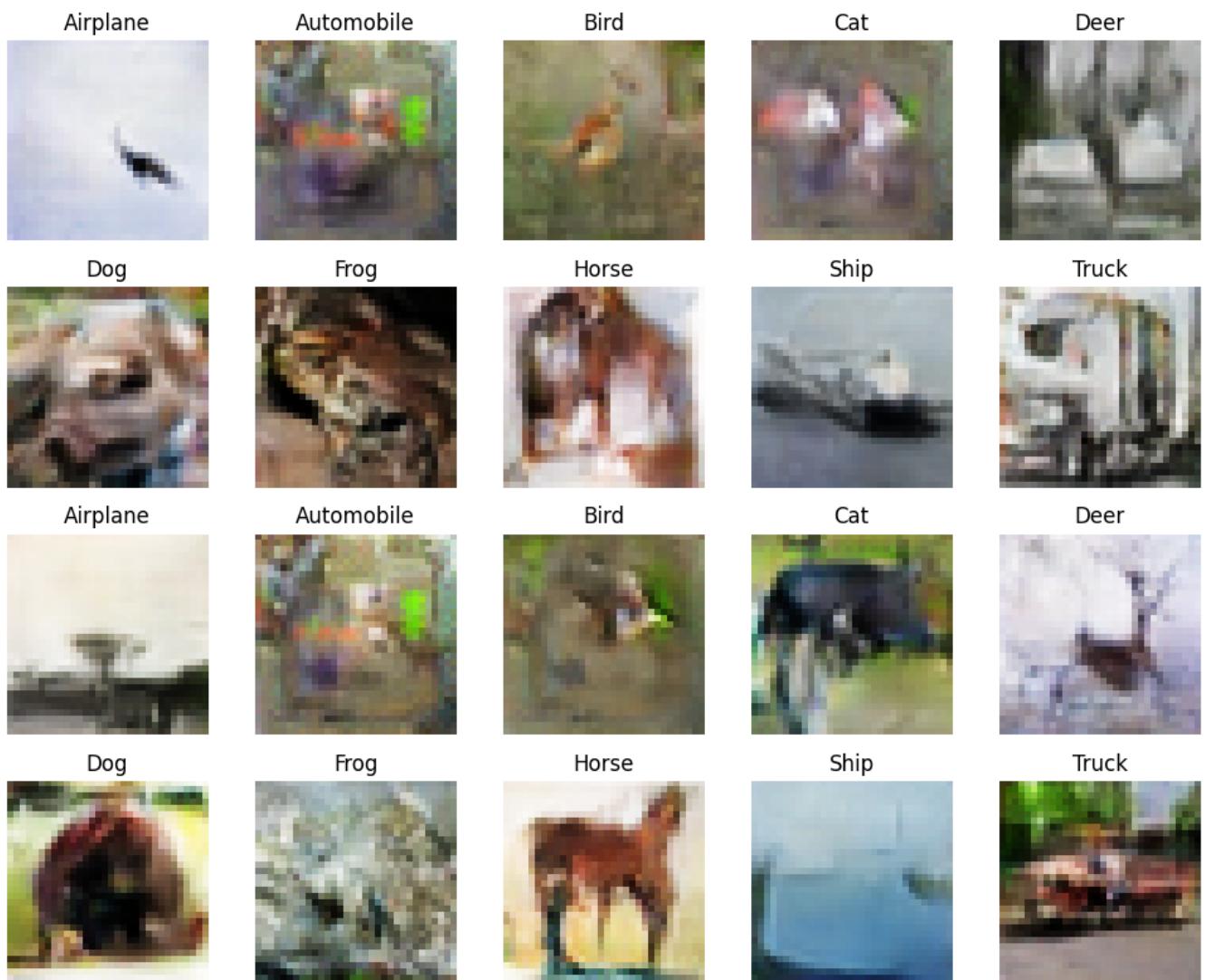
Epoch 164/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0744 - g_loss: 8.0729 - D(x|y): 0.4995 - D(G(z|y)): 0.0350 - KL Divergence: 4.7184

Epoch 165/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0700 - g_loss: 8.1394 - D(x|y): 0.5002 - D(G(z|y)): 0.0356 - KL Divergence: 4.4522

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - New ACGAN/generator-epoch-165.h5

Epoch 166/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0733 - g_loss: 8.0846 - D(x|y): 0.5002 - D(G(z|y)): 0.0344 - KL Divergence: 4.6408

Epoch 167/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0762 - g_loss: 7.9897 - D(x|y): 0.4999 - D(G(z|y)): 0.0364 - KL Divergence: 4.4707

Epoch 168/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0742 - g_loss: 8.0802 - D(x|y): 0.5002 - D(G(z|y)): 0.0366 - KL Divergence: 4.7756

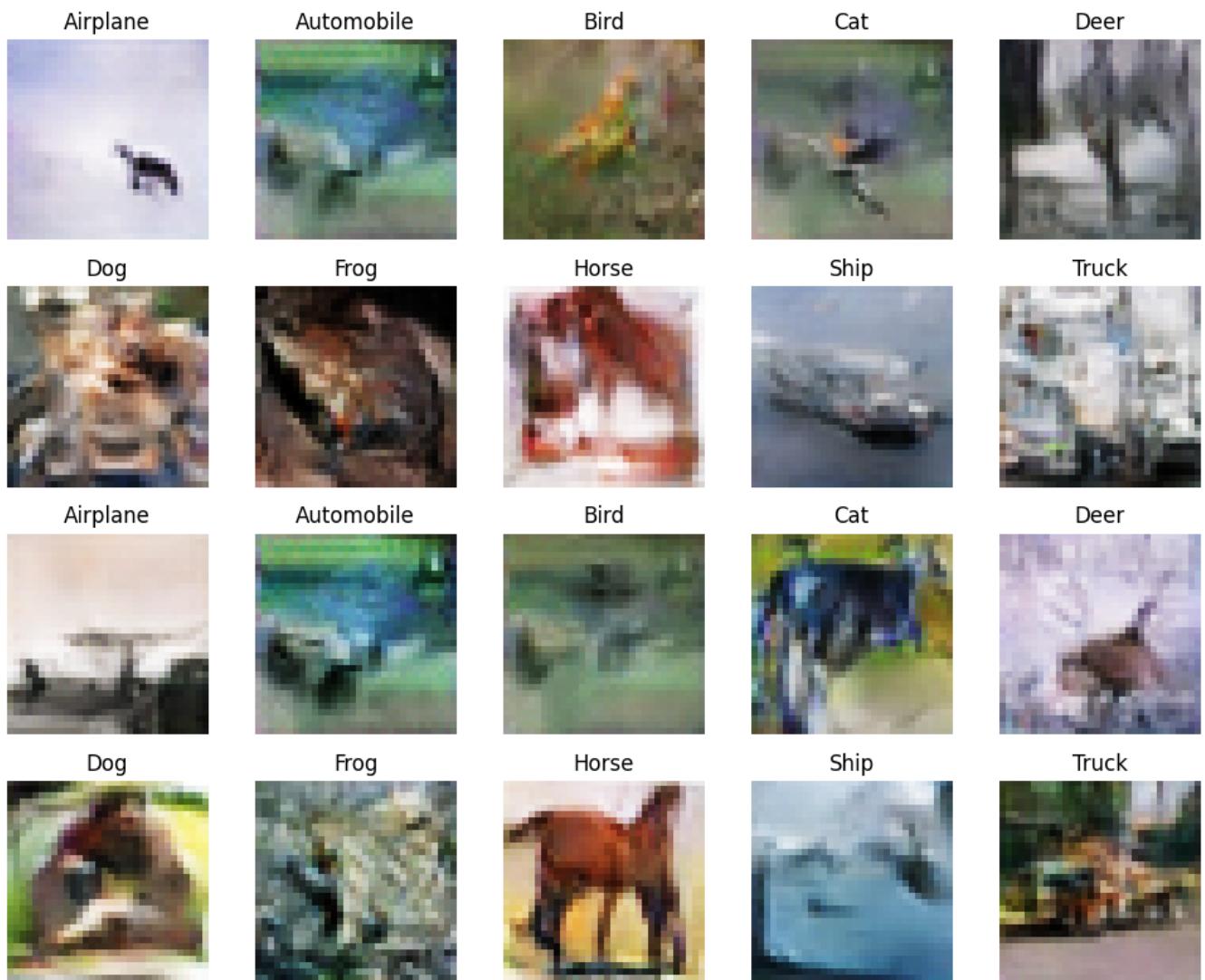
Epoch 169/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0728 - g_loss: 8.1807 - D(x|y): 0.5003 - D(G(z|y)): 0.0352 - KL Divergence: 4.4246

Epoch 170/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0724 - g_loss: 8.2117 - D(x|y): 0.5001 - D(G(z|y)): 0.0345 - KL Divergence: 4.8264

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New ACGAN/generator-epoch-170.h5

Epoch 171/200

782/782 [=====] - 88s 113ms/step - d_loss: 0.0755 - g_loss: 8.0926 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0378 - KL Divergence: 4.7642

Epoch 172/200

782/782 [=====] - 87s 112ms/step - d_loss: 0.0702 - g_loss: 8.1292 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0351 - KL Divergence: 4.3624

Epoch 173/200

782/782 [=====] - 87s 111ms/step - d_loss: 0.0696 - g_loss: 8.1703 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0334 - KL Divergence: 4.3526

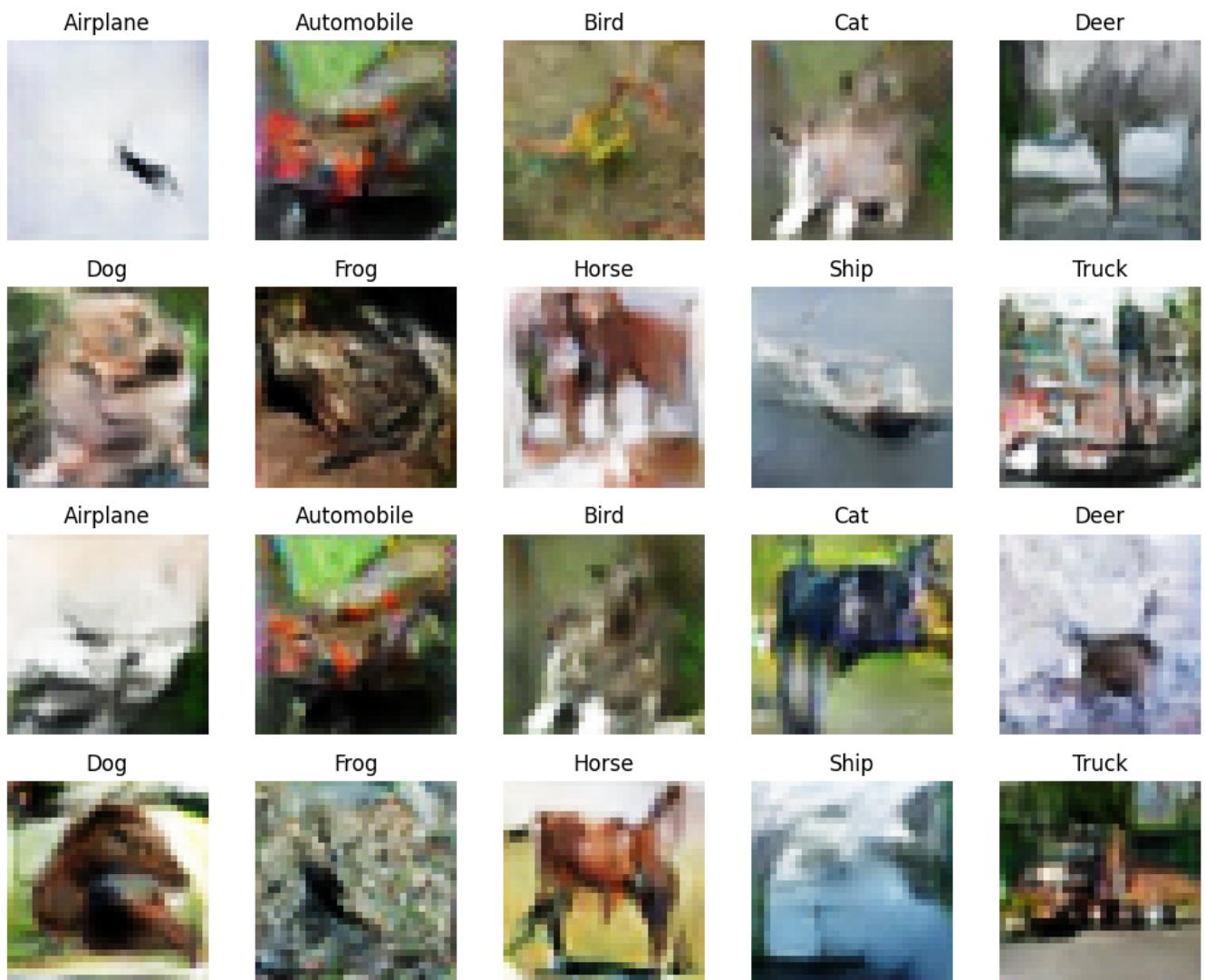
Epoch 174/200

782/782 [=====] - 87s 112ms/step - d_loss: 0.0684 - g_loss: 8.3260 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0353 - KL Divergence: 4.8449

Epoch 175/200

782/782 [=====] - 87s 111ms/step - d_loss: 0.0667 - g_loss: 8.0661 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0348 - KL Divergence: 4.6369

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New ACGAN/generator-epoch-175.h5

Epoch 176/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0651 - g_loss: 8.2996 - D(x|y): 0.5001 - D(G(z|y)): 0.0316 - KL Divergence: 4.8388

Epoch 177/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0716 - g_loss: 8.3022 - D(x|y): 0.5009 - D(G(z|y)): 0.0341 - KL Divergence: 4.5590

Epoch 178/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0660 - g_loss: 8.2865 - D(x|y): 0.5004 - D(G(z|y)): 0.0314 - KL Divergence: 4.6032

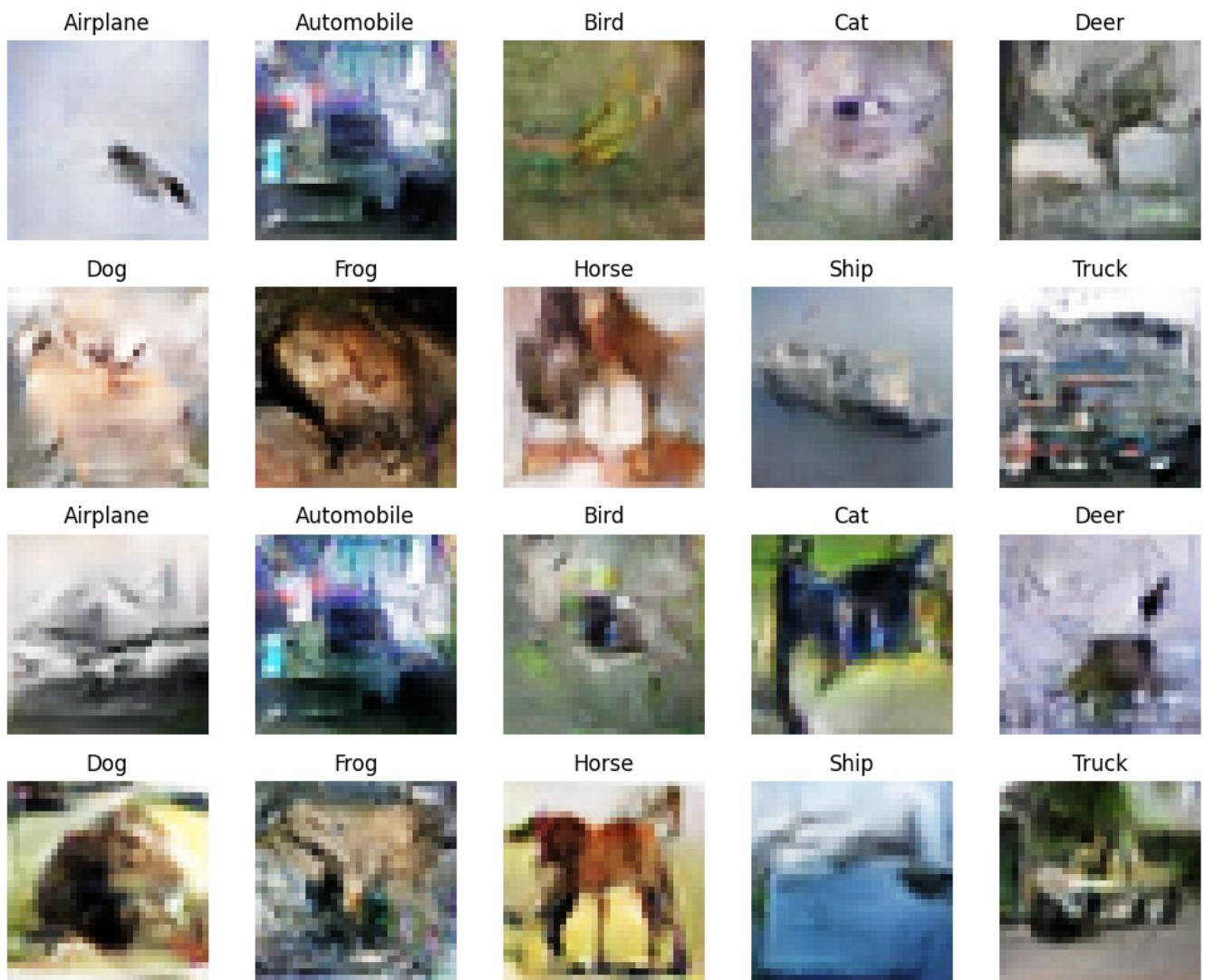
Epoch 179/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0684 - g_loss: 8.3598 - D(x|y): 0.5005 - D(G(z|y)): 0.0325 - KL Divergence: 4.5246

Epoch 180/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.0710 - g_loss: 8.2936 - D(x|y): 0.5002 - D(G(z|y)): 0.0319 - KL Divergence: 4.5682

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-180.h5

Epoch 181/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0671 - g_loss: 8.3628 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0321 - KL Divergence: 4.2278

Epoch 182/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0645 - g_loss: 8.5127 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0295 - KL Divergence: 4.5485

Epoch 183/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0649 - g_loss: 8.6284 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0297 - KL Divergence: 4.0881

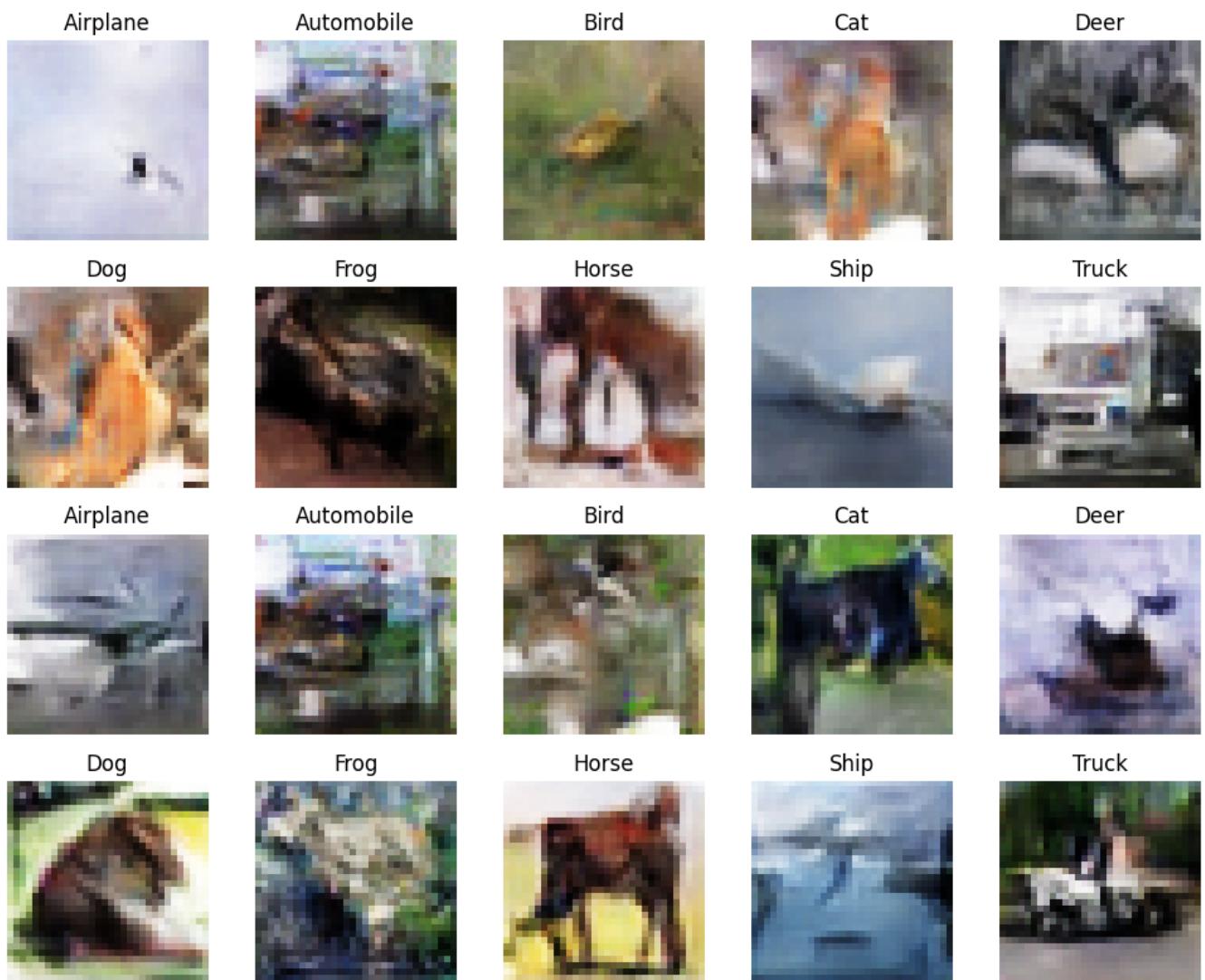
Epoch 184/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0615 - g_loss: 8.4513 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0286 - KL Divergence: 4.2055

Epoch 185/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0615 - g_loss: 8.6233 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0287 - KL Divergence: 4.5552

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New ACGAN/generator-epoch-185.h5

Epoch 186/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0641 - g_loss: 8.5929 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0293 - KL Divergence: 4.4563

Epoch 187/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0657 - g_loss: 8.5711 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0279 - KL Divergence: 4.4411

Epoch 188/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0685 - g_loss: 8.3770 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0295 - KL Divergence: 4.4341

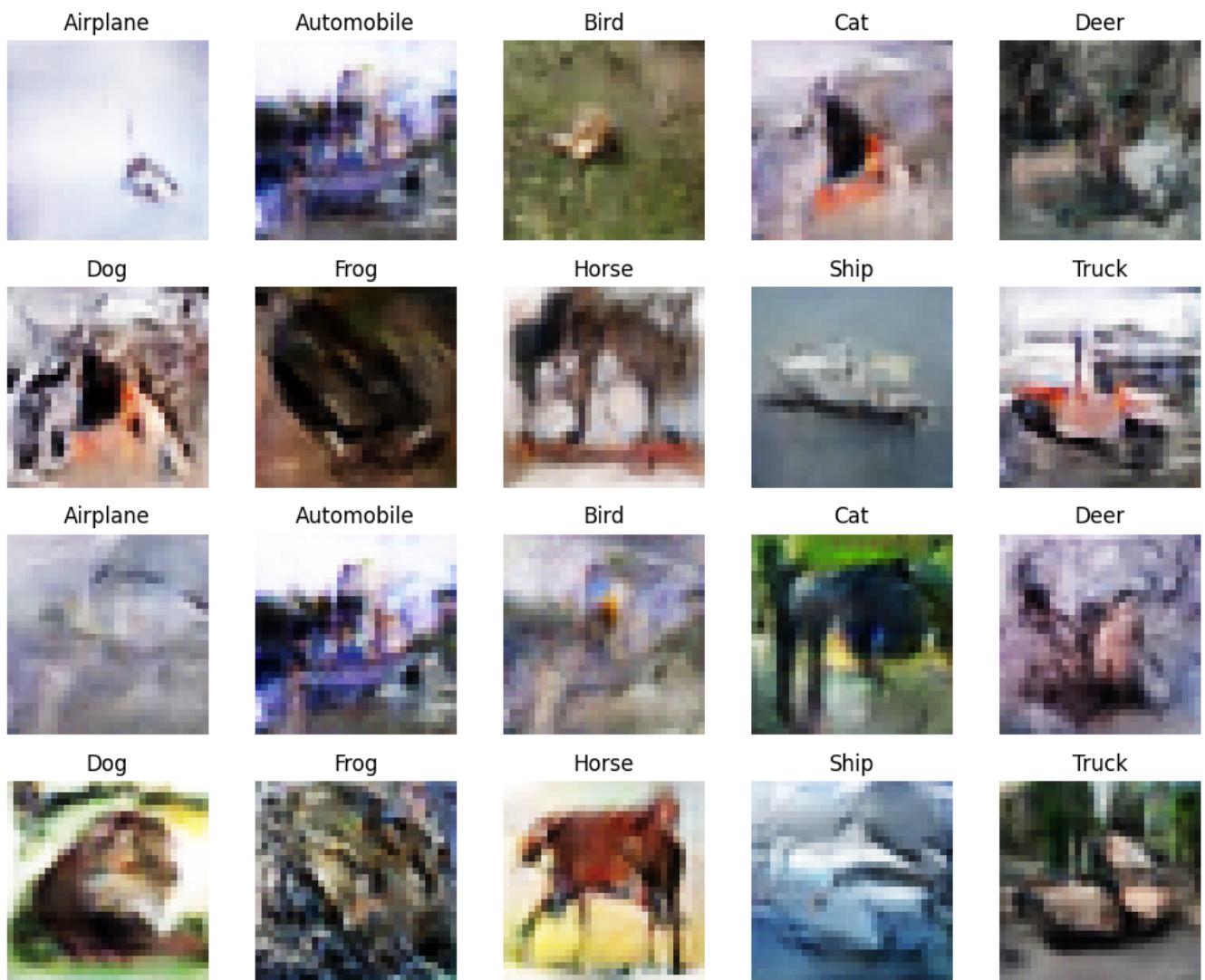
Epoch 189/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0608 - g_loss: 8.3988 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0268 - KL Divergence: 4.0901

Epoch 190/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0630 - g_loss: 8.7423 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0275 - KL Divergence: 4.4028

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New ACGAN/generator-epoch-190.h5

Epoch 191/200

782/782 [=====] - 86s 110ms/step - d_loss: 0.0656 - g_loss: 8.4579 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0283 - KL Divergence: 4.5031

Epoch 192/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0614 - g_loss: 8.4956 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0284 - KL Divergence: 4.6834

Epoch 193/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0619 - g_loss: 8.4405 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0273 - KL Divergence: 4.4189

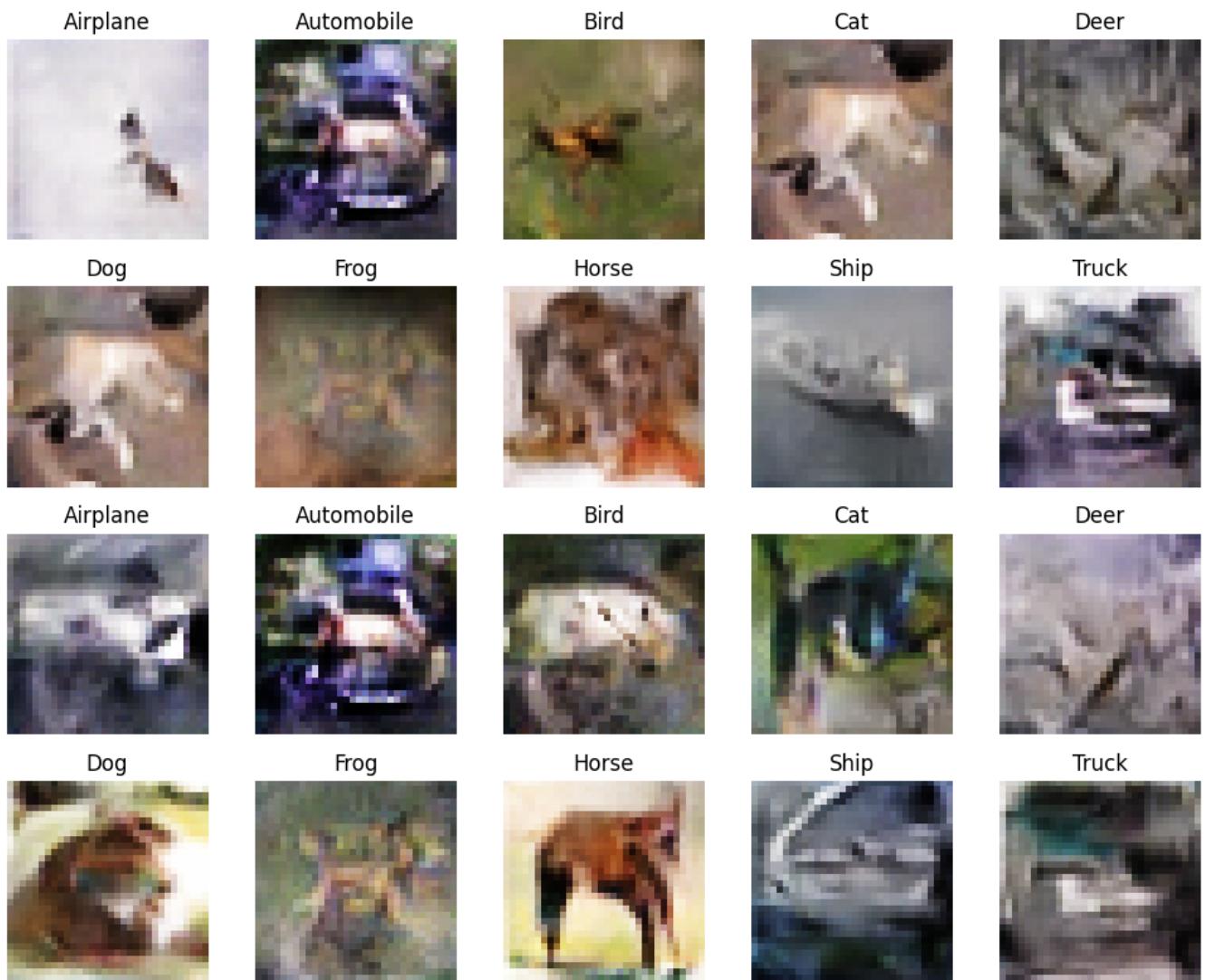
Epoch 194/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0604 - g_loss: 8.3121 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0268 - KL Divergence: 4.7363

Epoch 195/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0695 - g_loss: 8.4642 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0256 - KL Divergence: 4.8196

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New ACGAN/generator-epoch-195.h5

Epoch 196/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0769 - g_loss: 8.3449 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.0278$ - KL Divergence: 4.5927

Epoch 197/200

782/782 [=====] - 85s 109ms/step - d_loss: 0.0763 - g_loss: 7.9450 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0273$ - KL Divergence: 4.2954

Epoch 198/200

782/782 [=====] - 84s 108ms/step - d_loss: 0.0831 - g_loss: 8.2179 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.0301$ - KL Divergence: 4.7499

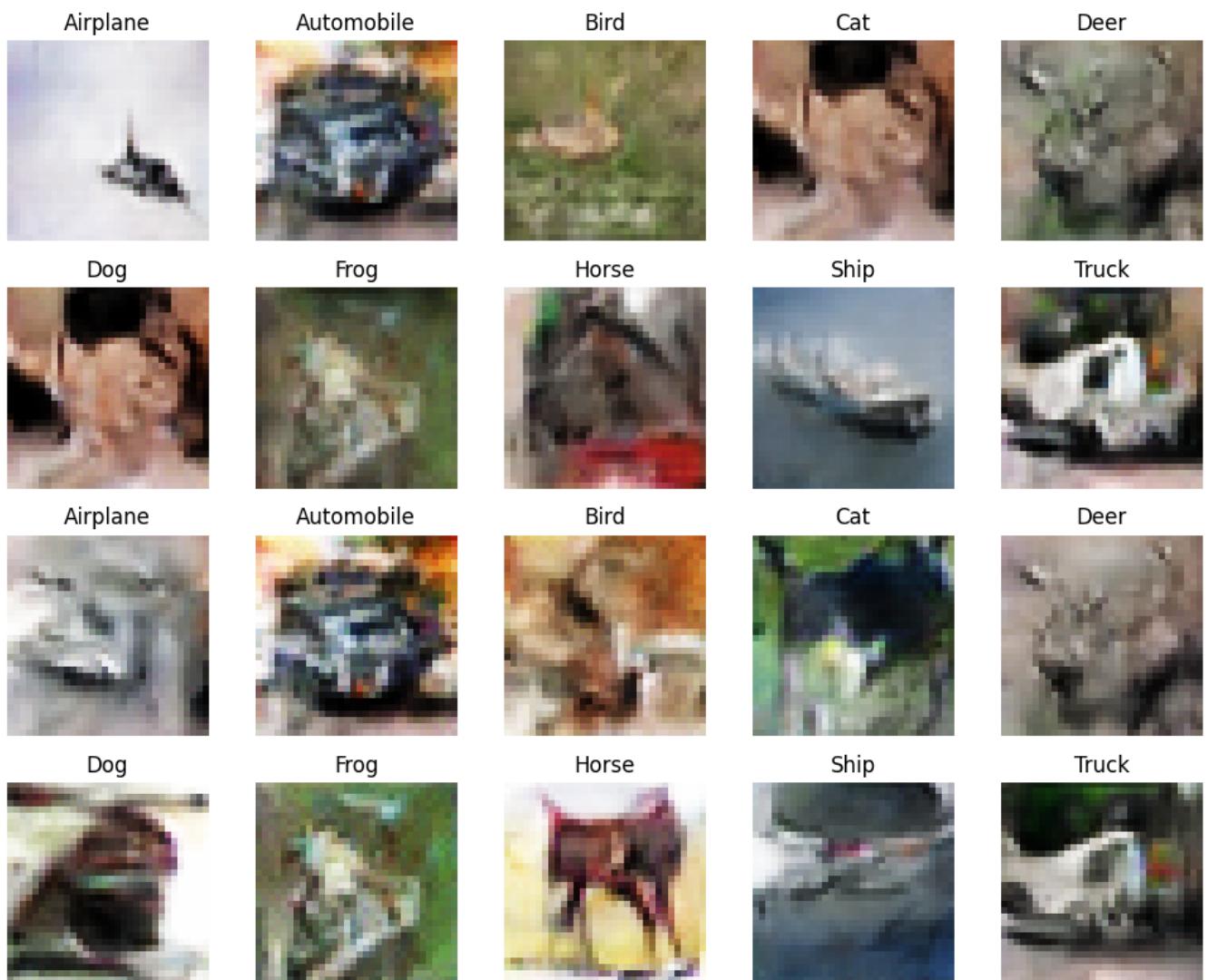
Epoch 199/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0884 - g_loss: 7.7769 -
 $D(x|y): 0.5007$ - $D(G(z|y)): 0.0325$ - KL Divergence: 4.7332

Epoch 200/200

782/782 [=====] - 85s 108ms/step - d_loss: 0.0805 - g_loss: 7.7837 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0294$ - KL Divergence: 4.5898

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New ACGAN/generator-epoch-Full Train.h5

Observations

We note that training the model, features like color etc was learnt by the model to generate the images. We also note that the images of some of the animals got blurred and fuzzy at the end. However we can see that unlike with the previous model giving the same image, the NewACGAN did not converge into a state for monotonous output. But we can see signs of the output becoming monotonous as the last epoch's trucks look quite similar. Suggesting that using Gaussian Weights and SpectralNormalisations to create the images reduce the changes of model collapsing but made the images generated more fuzzy.

A possible reason why the images are not as good and clear the previous ACGAN images is due to Gaussian weight leading to a slow convergence and poorer images generated. Spectral Normalisation just prevents the model from collapsing and generate more diverse images.

New ACGAN Evaluation

As mentioned previously we will be using a quantitative and manual evaluation of the ACGAN.

We will start off with a quantitative analysis so that we can find the best model to generate the images for manual evaluation.

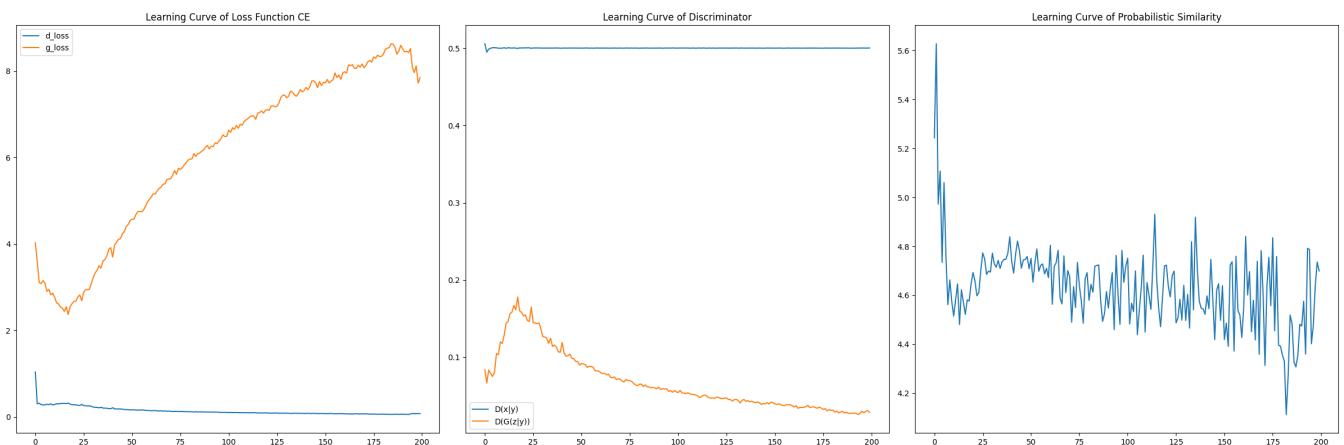
```
In [ ]: # store history object into dataframe
improve_aux_cond_hist_df = pd.DataFrame(improve_aux_cond_hist.history)

# using pandas dataframe to plot out Learning curve
fig, (ax1, ax2, ax3) = plt.subplots(
    1, 3, figsize=(24, 8), tight_layout=True)
```

```

improve_aux_cond_hist_df.loc[:, ["d_loss", 'g_loss']].plot(
    ax=ax1, title=r'Learning Curve of Loss Function CE')
improve_aux_cond_hist_df.loc[:, ['D(x|y)', 'D(G(z|y))']].plot(
    ax=ax2, title=r'Learning Curve of Discriminator')
improve_aux_cond_hist_df.loc[:, 'KL Divergence'].plot(
    ax=ax3, title=r'Learning Curve of Probabilistic Similarity')
plt.show()

```



Observations

We see that KL divergence has more spikes in training as the generator steady increases, this is likely due to the Gaussian Weight initialised making training slower. As 180 epochs is the lowest KL divergences, we will be using it as the best generator and show signs of model collapsing.

```
In [ ]: # Loading Weights for best Generator
saved_weights = 'New ACGAN\generator-epoch-180.h5'
improve_aux_cond_gan.generator.load_weights(saved_weights)
improve_aux_cond_gan.generator.summary()
```

Model: "generator_ACGAN"

Layer (type)	Output Shape	Param #	Connected to
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_3 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
=====			
Layer (type)	Output Shape	Param #	Connected to
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_3 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_3 (Dense)	(None, 2048)	284672	['concatenate_3[0][0]']
batch_normalization_12 (BatchNormalization)	(None, 2048)	8192	['dense_3[0][0]']
p_re_lu_4 (PReLU)	(None, 2048)	2048	['batch_normalization_12[0][0]']
reshape_1 (Reshape)	(None, 2, 2, 512)	0	['p_re_lu_4[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2784320	['reshape_1[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
=====			
Total params: 3,082,307			
Trainable params: 3,076,419			
Non-trainable params: 5,888			

In []:

```
n = 10000

# generating Labels
labels = np.random.randint(low=0, high=10, size=n)
one_hot_labels = to_categorical(labels)

# Generating 1000 Synthetic Images
random_noise = tf.random.normal(shape=(n, NOISE))

synthetic_images = improve_aux_cond_gan.generator.predict(
    [random_noise, one_hot_labels])
print("Latent Vector Dim: {} \t Generated Images Dim: {}".format(
    random_noise.shape, synthetic_images.shape))

# Scaling back to [0, 1]
synthetic_images -= -1
```

```

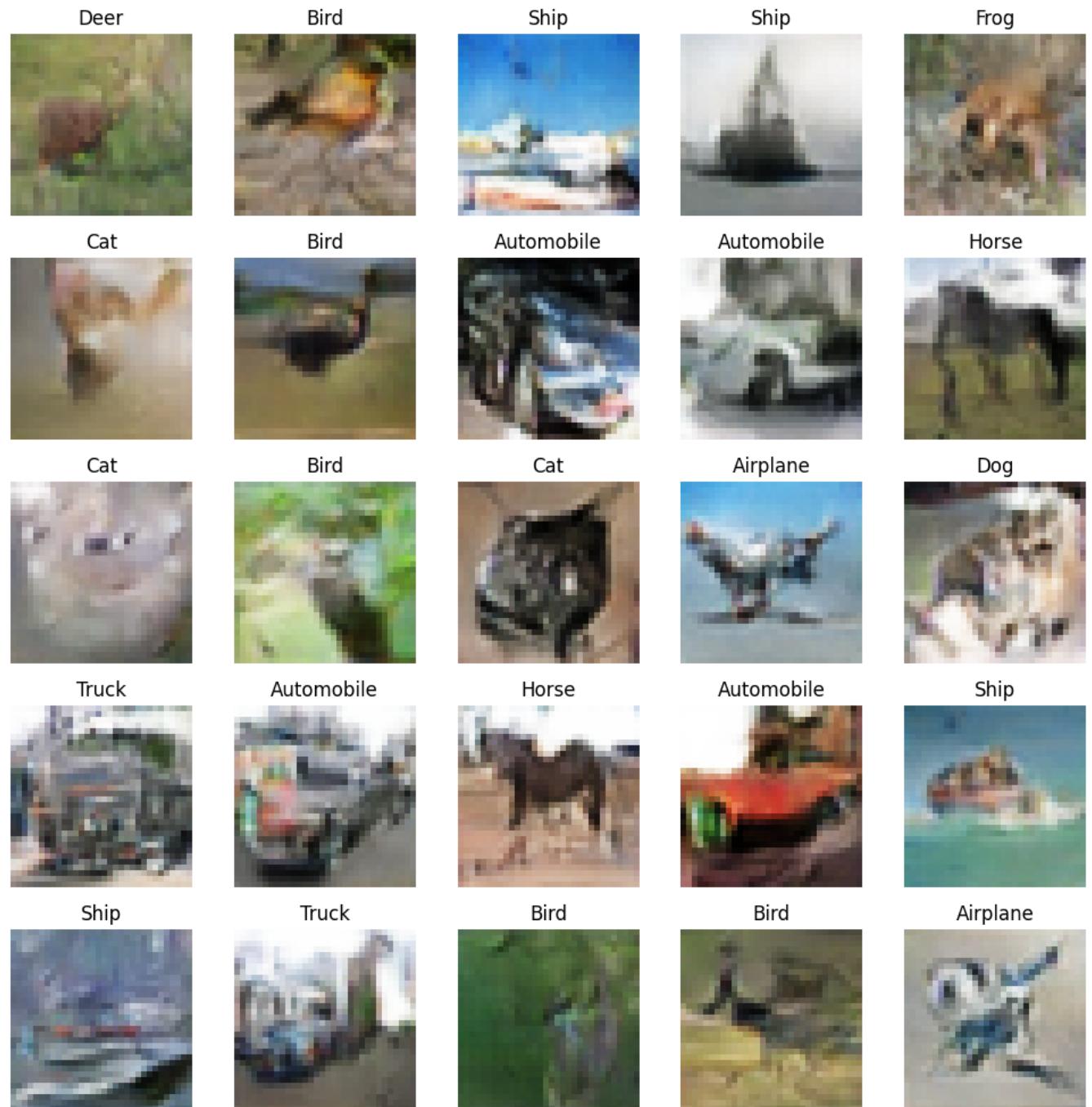
synthetic_images /= (1 - (-1))

# Display 25 randomly sampled images
fig = plt.figure(figsize=(10, 10), tight_layout=True)
for i in range(25):
    rand_idx = np.random.randint(0, len(synthetic_images))
    ax = fig.add_subplot(5, 5, i+1)
    ax.imshow(synthetic_images[rand_idx])
    ax.set_title(class_labels[labels[rand_idx]])
    ax.axis('off')
plt.show()

```

313/313 [=====] - 6s 20ms/step

Latent Vector Dim: (10000, 128) Generated Images Dim: (10000, 32, 32, 3)



Observations

We see that the shapes of the animals are better but there are certain images where the images are very fuzzy and blurry.

```
In [ ]: improve_aux_cond_gan_fid_class = GAN_FID(batch_size=512, noise=128,
                                              sample_size=10000, buffer_size=1024, labels=True)
improve_aux_cond_gan_fid_class.fit()
```

```
generator=improve_aux_cond_gan.generator, train_data=x_train)
fid_score = improve_aux_cond_gan_fid_class.evaluate()
```

Computing Real Image Embeddings

```
0%|          | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 8s 153ms/step
16/16 [=====] - 2s 155ms/step
16/16 [=====] - 2s 156ms/step
16/16 [=====] - 3s 163ms/step
16/16 [=====] - 2s 136ms/step
16/16 [=====] - 2s 136ms/step
16/16 [=====] - 2s 155ms/step
16/16 [=====] - 2s 157ms/step
16/16 [=====] - 2s 154ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 134ms/step
```

Computing Generated Image Embeddings

```
0%|          | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 135ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 134ms/step
16/16 [=====] - 2s 135ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 135ms/step
16/16 [=====] - 2s 134ms/step
16/16 [=====] - 2s 136ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 133ms/step
16/16 [=====] - 2s 135ms/step
16/16 [=====] - 2s 138ms/step
16/16 [=====] - 2s 139ms/step
16/16 [=====] - 2s 135ms/step
```

Computed Embeddings Real Images Embedding Shape: (10240, 2048) Generated Images Embedding Shape: (10240, 2048)

The computed FID score is: 106.80945871926016

Observations

We see the FID score is higher than the original ACGAN model from 88.7613090961658 to 106.80945871926016. This suggest that the changes made did not improve the model. As mentioned previously, Gaussian weights make the model train more steadily but with a slower speed.

cGAN

New cGAN Discriminator

As mentioned in the DCGAN discriminator creation, we will be using Gaussian Weights for the new ACGAN model.

Gaussian weights will reduce the instability of training [mean = 0, std = 0.02].

```
In [ ]: # function to create discriminator model
def create_improve_cGAN_discriminator(image_size):
    # input image
    img_input = Input(shape=(image_size), name='Image_Input')
    weights_init = RandomNormal(mean=0, stddev=0.02)

    # conditions y
    conditions = Input(shape=(10,), name='Conditions_y')

    base_discriminator = Sequential([
        Conv2D(64, kernel_size=4, strides=2, padding='same',
               kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        Conv2D(128, kernel_size=4, strides=2, padding='same',
               kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        Conv2D(256, kernel_size=4, strides=2, padding='same',
               kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
        Conv2D(512, kernel_size=4, strides=2, padding='same',
               kernel_initializer=weights_init),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),
    ], name='Base_Discriminator')
    discriminator = base_discriminator(img_input)

    discriminator = GlobalAveragePooling2D()(discriminator)

    # Concatenate - combine with conditions y
    merged_layer = Concatenate()([discriminator, conditions])
    discriminator = Dense(512, activation='relu')(merged_layer)

    # Output
    discriminator = Dense(1, activation='sigmoid',
                          name='Output_Layer')(discriminator)

    discriminator = Model(inputs=[img_input, conditions],
                          outputs=discriminator, name='discriminator_cGAN')
    return discriminator

create_improve_cGAN_discriminator(image_size=IMG_SIZE).summary()
```

Model: "discriminator_cGAN"

Layer (type)	Output Shape	Param #	Connected to
=====			
Image_Input (InputLayer)	[(None, 32, 32, 3)]	0	[]
Base_Discriminator (Sequential)	(None, 2, 2, 512)	2760384	['Image_Input[0][0]']
global_average_pooling2d_4 (G1)	(None, 512)	0	['Base_Discriminator[0][0]']
Conditions_y (InputLayer)	[(None, 10)]	0	[]
concatenate_7 (Concatenate)	(None, 522)	0	['global_average_pooling2d_4[0][0]', 'Conditions_y[0][0]']
dense_5 (Dense)	(None, 512)	267776	['concatenate_7[0][0]']
Output_Layer (Dense)	(None, 1)	513	['dense_5[0][0]']
=====			

```
c:\Users\Soh Hong Yu\anaconda3\envs\gpu_env\lib\site-packages\keras\initializers\initializers_v2.py:120: UserWarning: The initializer RandomNormal is unseeded and being called multiple times, which will return identical values each time (even if the initializer is unseeded). Please update your code to provide a seed to the initializer, or avoid using the same initializer instance more than once.
```

```
    warnings.warn(
```

```
Total params: 3,028,673
```

```
Trainable params: 3,026,753
```

```
Non-trainable params: 1,920
```

```
Trainable params: 3,026,753
```

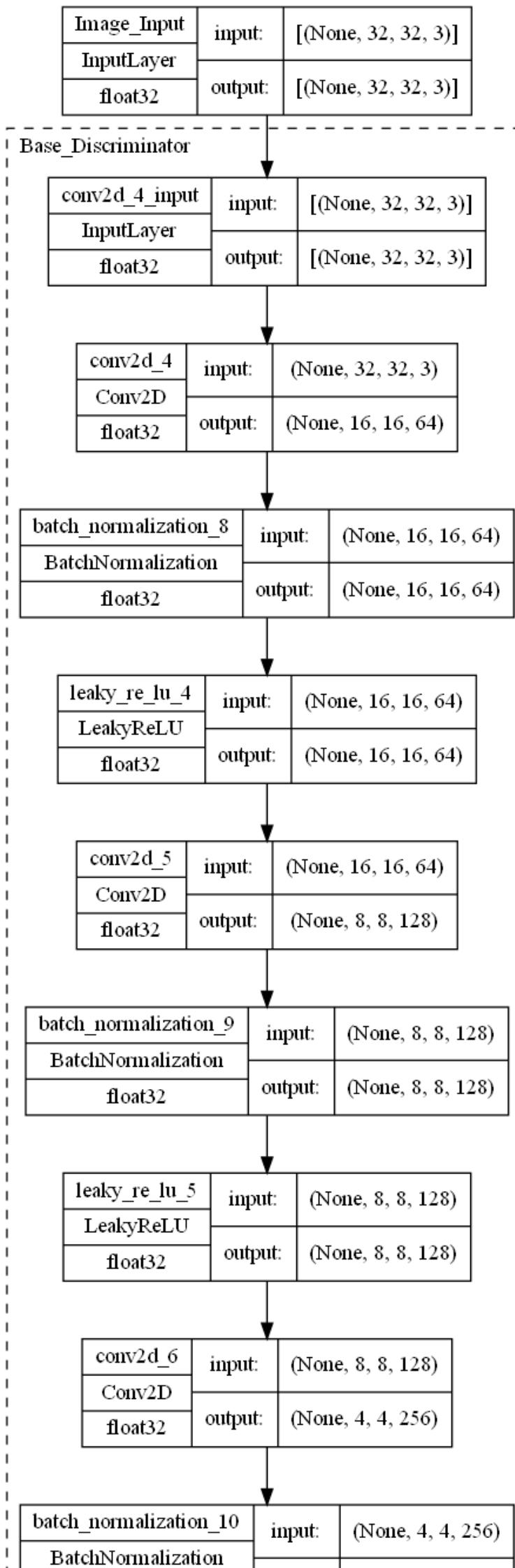
```
Non-trainable params: 1,920
```

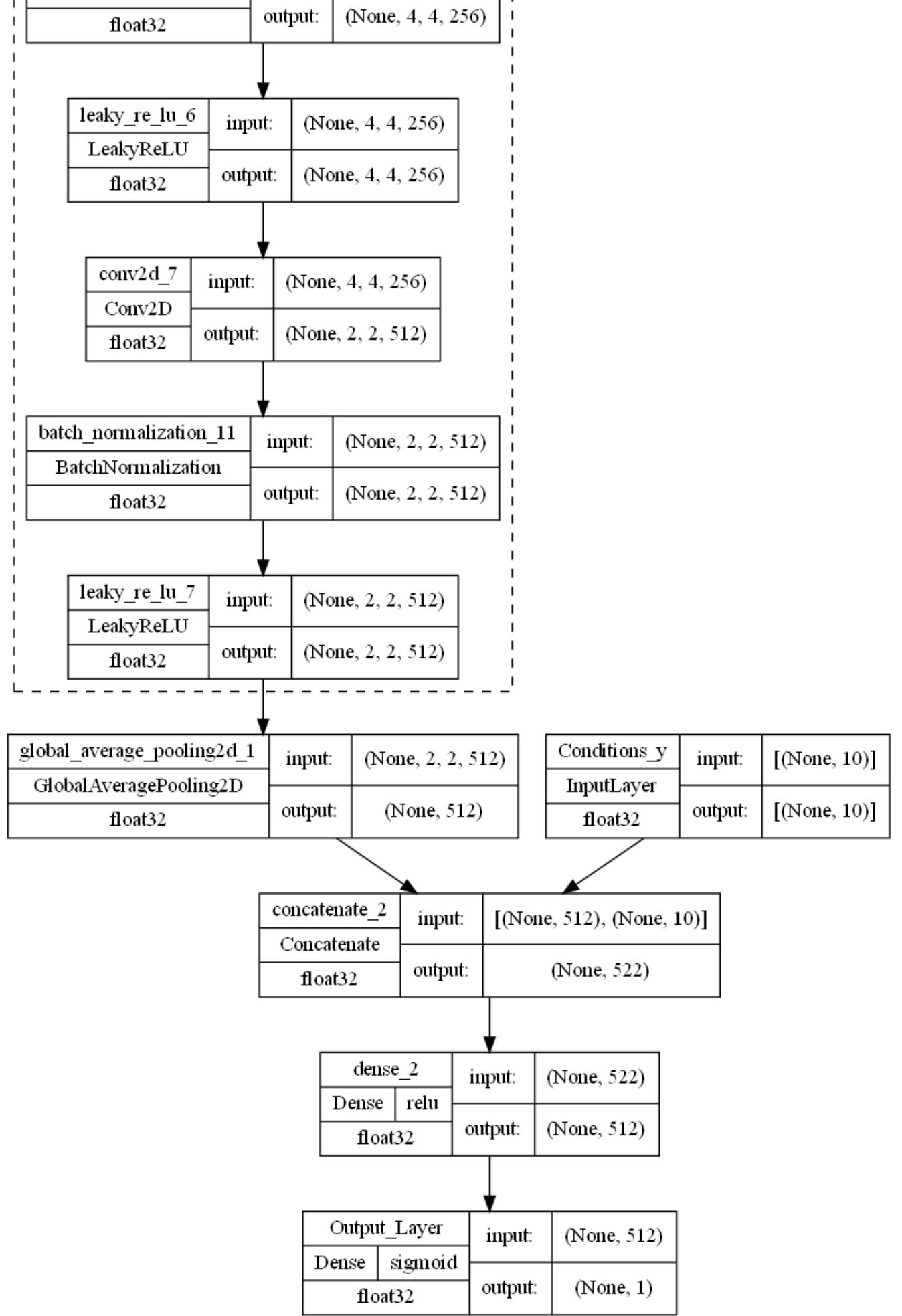
```
In [ ]:
```

```
plot_model(create_improve_cGAN_discriminator(image_size=IMG_SIZE), to_file="images/models/New
```

```
show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True
```

Out[]:





New cGAN Generator

Similarly, we will use Spectral Normalisation, Gaussian weights and PReLU for the generator. This will allow the generator to adapt better and learn better overall compared to the LeakyReLU used previously.

In []: # function to create generator model

```
def create_improve_cGAN_generator(noise):

    # gaussian weights initialization
    weights_init = RandomNormal(mean=0, stddev=0.02)

    # Latent noise vector z
    z = Input(shape=(noise,), name="Latent_Noise_Vector_z")

    # conditions y
    conditions = Input(shape=(10,), name='Conditions_y')

    # Generator network
    merged_layer = Concatenate()([z, conditions])

    # FC: 2x2x512
    generator = Dense(2*2*512, activation='relu')(merged_layer)
    generator = BatchNormalization(momentum=0.8)(generator)
    generator = PReLU()(generator)
    generator = Reshape((2, 2, 512))(generator)

    base_generator = Sequential([
        # Conv 1: 4x4x256
        SpectralNormalization(Conv2DTranspose(256, kernel_size=4, strides=2,
                                             padding='same', kernel_initializer=weights_init),
                             BatchNormalization(momentum=0.8),
                             PReLU(),
        # Conv 2: 8x8x128
        SpectralNormalization(Conv2DTranspose(128, kernel_size=4, strides=2,
                                             padding='same', kernel_initializer=weights_init),
                             BatchNormalization(momentum=0.8),
                             PReLU(),
        # Conv 3: 16x16x64
        SpectralNormalization(Conv2DTranspose(64, kernel_size=4, strides=2,
                                             padding='same', kernel_initializer=weights_init),
                             BatchNormalization(momentum=0.8),
                             PReLU(),
        ], name='Base_Generator')
    generator = base_generator(generator)

    # Conv 4: 32x32x3
    generator = Conv2DTranspose(3, kernel_size=4, strides=2, padding='same',
                               activation='tanh', name='Output_Layer')(generator)

    generator = Model(inputs=[z, conditions],
                      outputs=generator, name='generator_cGAN')
    return generator

create_improve_cGAN_generator(noise=NOISE).summary()
```

Model: "generator_cGAN"

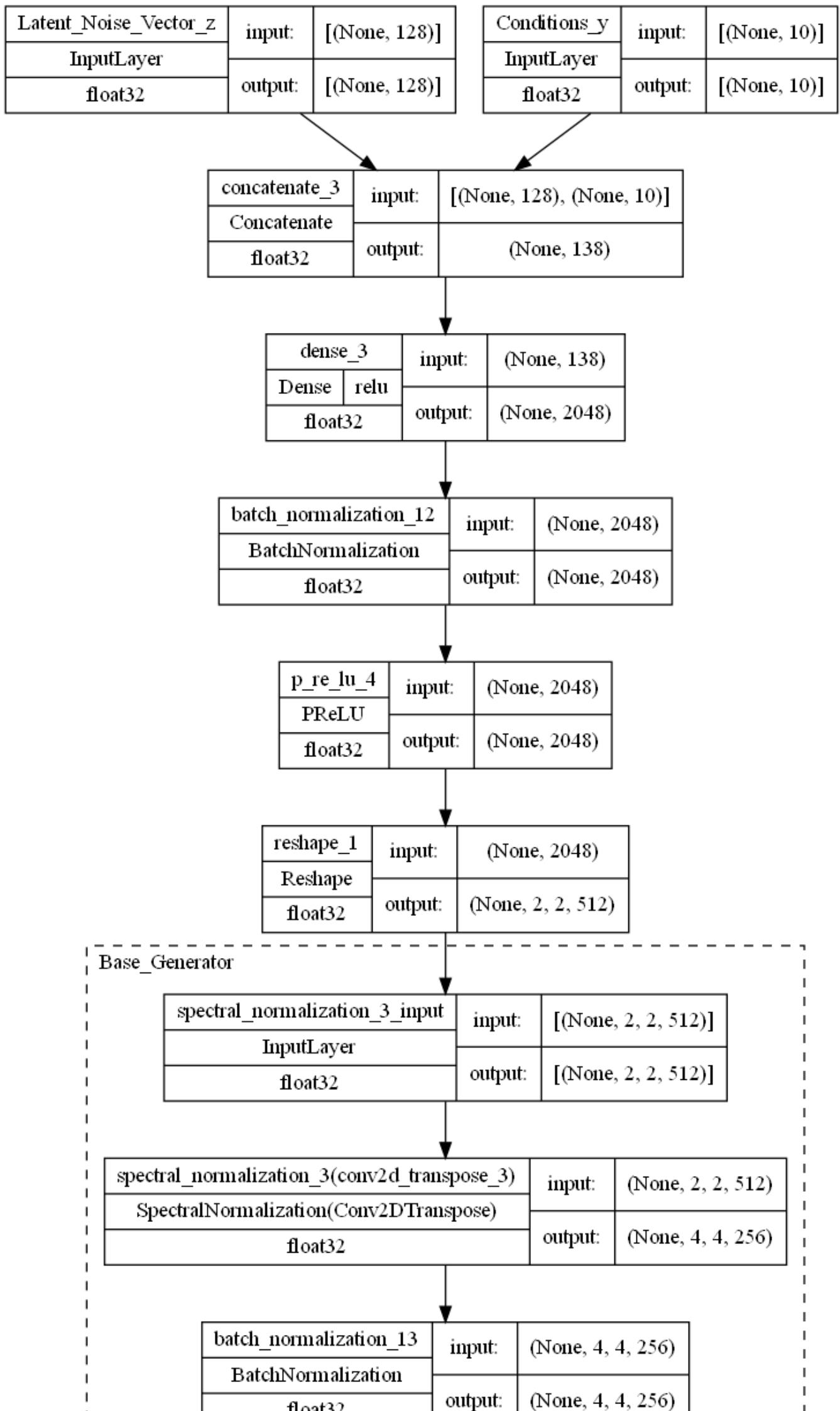
Layer (type)	Output Shape	Param #	Connected to
<hr/>			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_8 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_6 (Dense)	(None, 2048)	284672	['concatenate_8[0][0]']

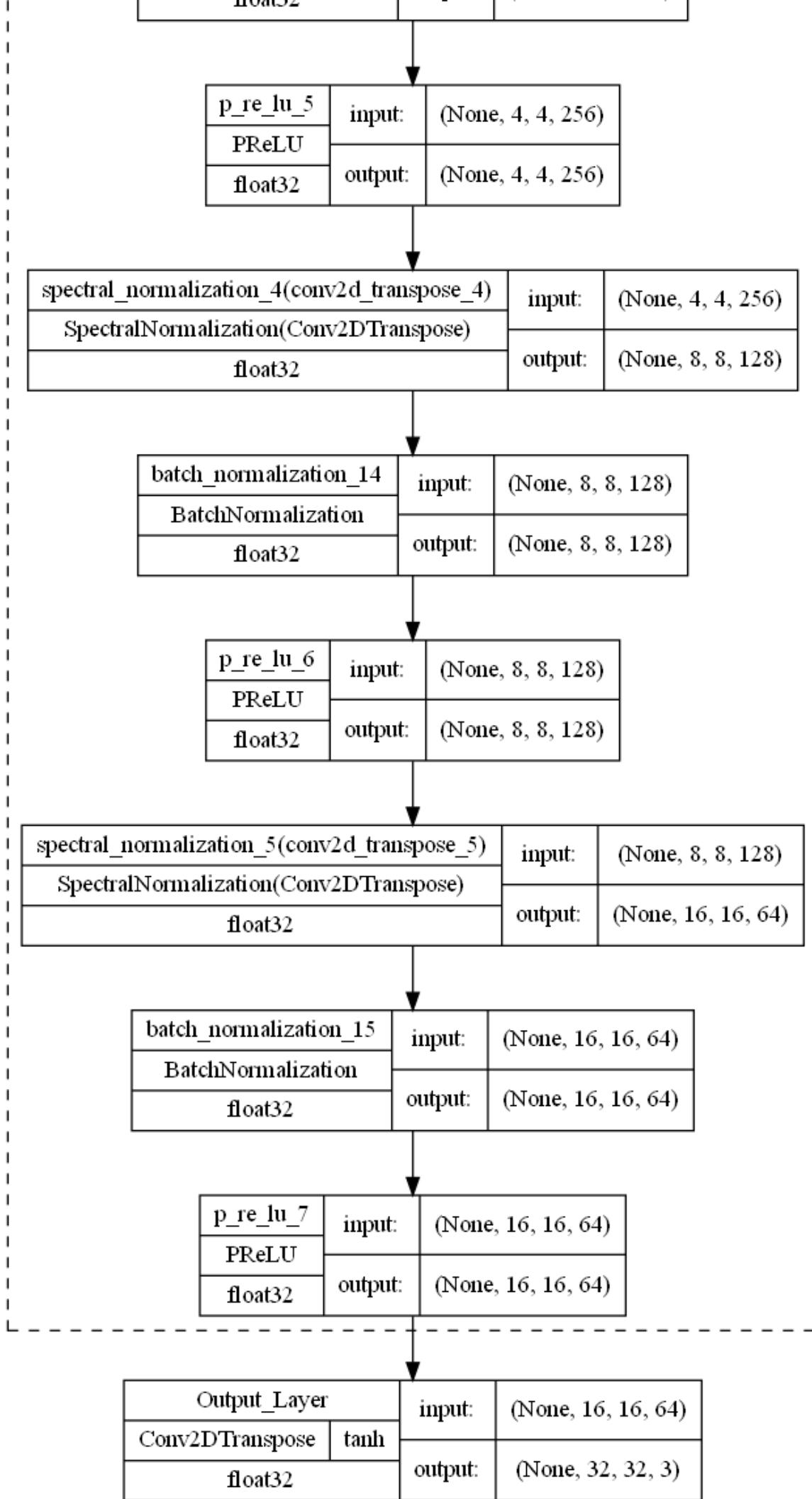
Layer (type)	Output Shape	Param #	Connected to
<hr/>			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_8 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_6 (Dense)	(None, 2048)	284672	['concatenate_8[0][0]']
batch_normalization_118 (Batch Normalization)	(None, 2048)	8192	['dense_6[0][0]']
p_re_lu_8 (PReLU)	(None, 2048)	2048	['batch_normalization_118[0][0]']
reshape_2 (Reshape)	(None, 2, 2, 512)	0	['p_re_lu_8[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2784320	['reshape_2[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']

Total params: 3,082,307
Trainable params: 3,076,419
Non-trainable params: 5,888

```
In [ ]: plot_model(create_improve_cGAN_generator(noise=NOISE), to_file="images/models/New cGAN_generator", show_shapes=True, show_dtype=True, expand_nested=True, show_layer_activations=True)
```

Out[]:





```
In [ ]: class NewcGANMonitor(keras.callbacks.Callback):
    def __init__(self, num_img=20, noise=128, patience=10, vmin=0, vmax=1):
        self.num_img = num_img
        self.noise = noise
        self.patience = patience
        self.vmin = vmin
        self.vmax = vmax
        self.latent_noise_vector = tf.random.normal(
            shape=(self.num_img, self.noise))
        self.conditions = to_categorical([0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
                                         0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

    def generate_plot(self):
        # Generate Images
        generated_images = self.model.generator.predict(
            [self.latent_noise_vector, self.conditions])
        # Normalise Image from [vmin, vmax] to [0, 1]
        generated_images -= self.vmin
        generated_images /= (self.vmax - self.vmin)
        row_size = int(np.ceil(self.num_img/5))
        fig = plt.figure(figsize=(10, 2*row_size), tight_layout=True)
        for i in range(self.num_img):
            ax = fig.add_subplot(row_size, 5, i+1)
            ax.imshow(generated_images[i])
            ax.set_title(class_labels[i % 10])
            ax.axis('off')
        plt.show()

    def save_weights(self, epoch=None):
        try:
            if epoch != None:
                name = 'New cGAN/generator-{}.h5'.format(epoch)
                print('Generator Checkpoint - {}'.format(name))
                self.model.generator.save_weights(
                    filepath=name,
                    save_format='h5'
                )
        except Exception as e:
            print(e)

    def on_epoch_begin(self, epoch, logs=None):
        if epoch % self.patience == 0:
            self.generate_plot()
            self.save_weights(epoch)

    def on_train_end(self, epoch, logs=None):
        self.generate_plot()
        self.save_weights('Full Train')
```

```
In [ ]: callbacks = [
    NewcGANMonitor(num_img=20, noise=128, patience=5, vmin=-1, vmax=1),
]
```

New cGAN Training

```
In [ ]: class ConditionalGAN(tf.keras.Model):
    def __init__(self, discriminator, generator, noise):
        super(ConditionalGAN, self).__init__()
        self.discriminator = discriminator
        self.generator = generator
        self.noise = noise
        self.gen_loss_tracker = tf.keras.metrics.Mean(name="generator_loss")
        self.disc_loss_tracker = tf.keras.metrics.Mean(
            name="discriminator_loss")
```

```

self.d_xy_tracker = tf.keras.metrics.Mean(name='Mean D(x|y)')
self.d_g_zy_tracker = tf.keras.metrics.Mean(name='Mean D(G(z|y))')
self.kl = tf.keras.metrics.KLDivergence()

def compile(self, d_optimizer, g_optimizer, loss_fn):
    super(ConditionalGAN, self).compile()
    self.d_optimizer = d_optimizer
    self.g_optimizer = g_optimizer
    self.loss_fn = loss_fn

def train_step(self, data):
    ### TRAINING DISCRIMINATOR ####
    # Unpack the data.
    real_images, condition = data

    # Sample for latent noise vector z
    batch_size = tf.shape(real_images)[0]
    latent_noise_vector = tf.random.normal(shape=(batch_size, self.noise))

    # Maps the noise Latent vector and Labels to generate fake images.
    generated_images = self.generator([latent_noise_vector, condition])

    # Combine with real images
    combined_images = tf.concat([generated_images, real_images], axis=0)
    combined_condition = tf.concat([condition, condition], axis=0)

    # Discrimination
    labels = tf.concat(
        [tf.zeros((batch_size, 1)), tf.ones((batch_size, 1))], axis=0
    )

    # Train the discriminator.
    with tf.GradientTape() as tape:
        first_predictions = self.discriminator(
            [combined_images, combined_condition])
        d_loss = self.loss_fn(labels, first_predictions)
        grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights)
        )

    # Computing D(x/y)
    d_xy = tf.math.reduce_mean(first_predictions)

    ### TRAINING GENERATOR ####
    latent_noise_vector = tf.random.normal(shape=(batch_size, self.noise))

    # Assemble labels that say "all real images".
    misleading_labels = tf.ones((batch_size, 1))

    with tf.GradientTape() as tape:
        fake_images = self.generator([latent_noise_vector, condition])
        second_predictions = self.discriminator([fake_images, condition])
        g_loss = self.loss_fn(misleading_labels, second_predictions)
        grads = tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(
            zip(grads, self.generator.trainable_weights))

    # Computing D(G(z/y))
    d_g_zy = tf.math.reduce_mean(second_predictions)

    # Monitor Loss and metrics.
    self.gen_loss_tracker.update_state(g_loss)
    self.disc_loss_tracker.update_state(d_loss)
    self.d_xy_tracker.update_state(d_xy)
    self.d_g_zy_tracker.update_state(d_g_zy)

```

```
self.kl.update_state(real_images, fake_images)

    return {
        "d_loss": self.disc_loss_tracker.result(),
        "g_loss": self.gen_loss_tracker.result(),
        "D(x|y)": self.d_xy_tracker.result(),
        "D(G(z|y))": self.d_g_zx_tracker.result(),
        "KL Divergence": self.
        kl.result(),
    }
```

```
In [ ]: tf.keras.backend.clear_session()
K.clear_session()

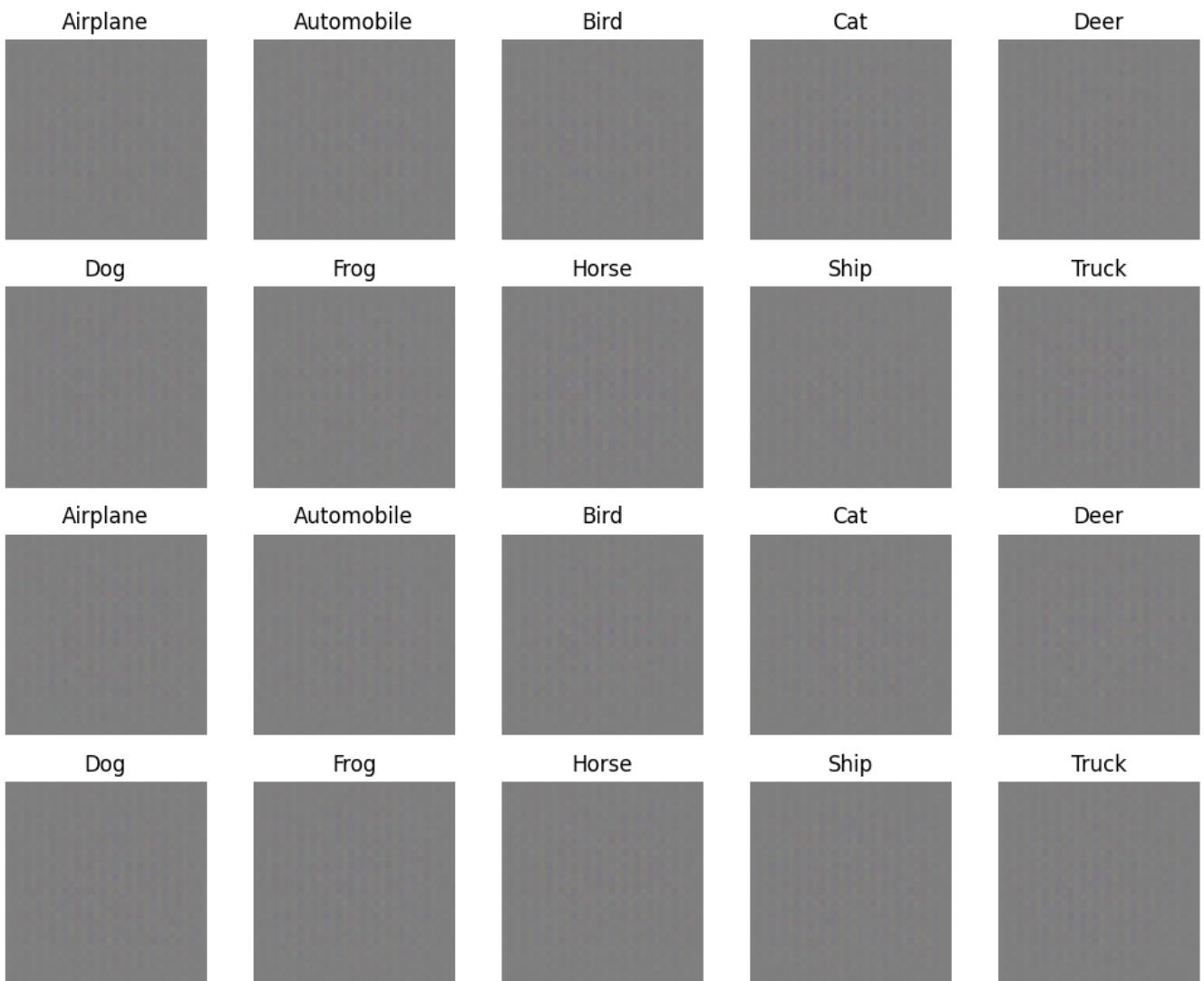
improve_cond_gan = ConditionalGAN(
    discriminator=create_improve_cGAN_discriminator(image_size=IMG_SIZE),
    generator=create_improve_cGAN_generator(noise=NOISE),
    noise=NOISE
)

improve_cond_gan.compile(
    d_optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    g_optimizer=keras.optimizers.Adam(learning_rate=0.0002, beta_1=0.5),
    loss_fn=keras.losses.BinaryCrossentropy(),
)

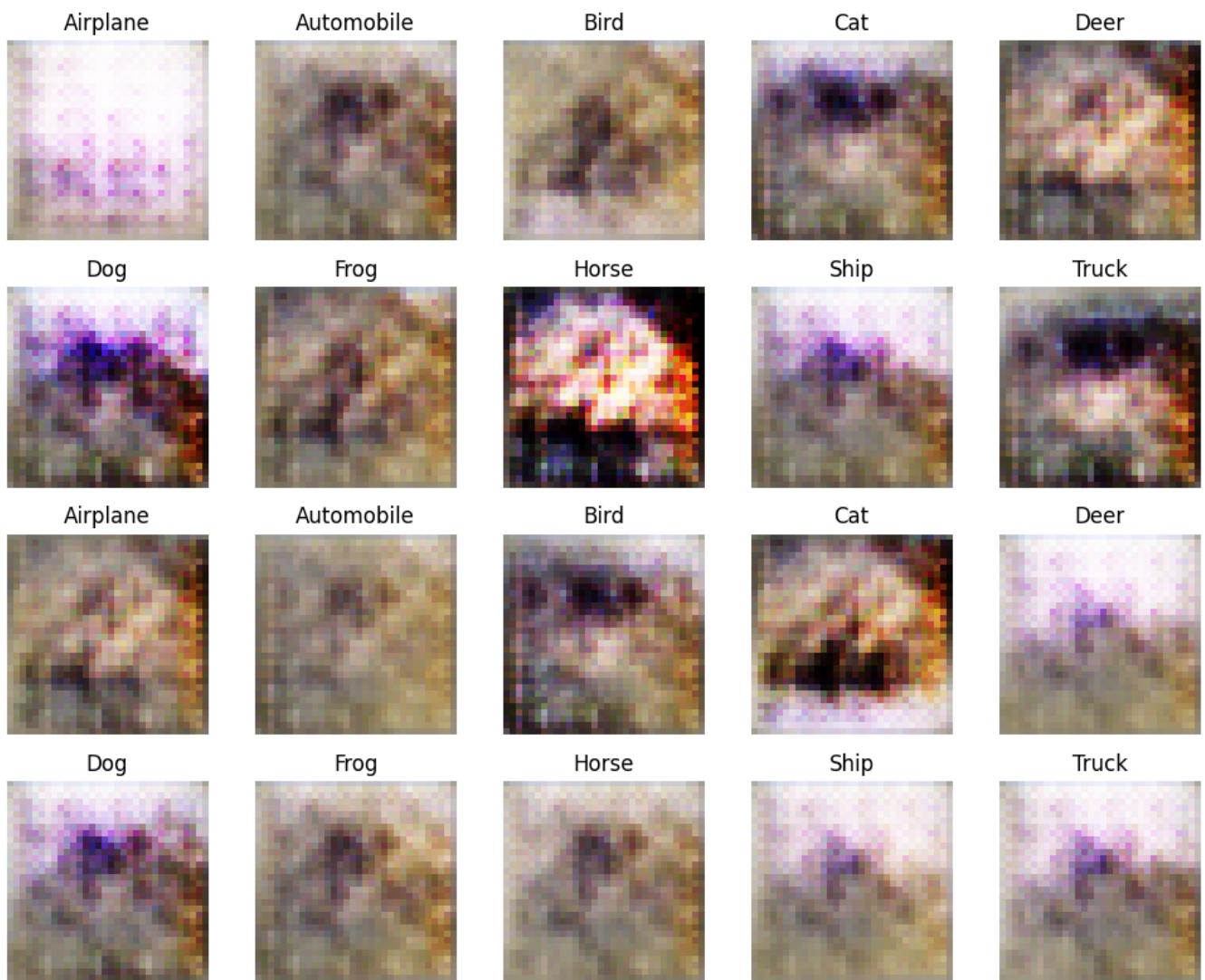
dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
dataset = dataset.shuffle(buffer_size=1024).batch(
    BATCH_SIZE, num_parallel_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)

improve_cond_gan_hist = improve_cond_gan.fit(
    dataset, epochs=200, use_multiprocessing=True, workers=16, callbacks=callbacks)
```

1/1 [=====] - 7s 7s/step



Generator Checkpoint - New cGAN/generator-epoch-0.h5
Epoch 1/200
782/782 [=====] - 54s 62ms/step - d_loss: 0.2978 - g_loss: 3.3459 -
D(x|y): 0.5221 - D(G(z|y)): 0.1088 - KL Divergence: 6.0988
Epoch 2/200
782/782 [=====] - 48s 62ms/step - d_loss: 0.2720 - g_loss: 3.0355 -
D(x|y): 0.4933 - D(G(z|y)): 0.0826 - KL Divergence: 6.7751
Epoch 3/200
782/782 [=====] - 46s 59ms/step - d_loss: 0.2494 - g_loss: 3.2739 -
D(x|y): 0.4964 - D(G(z|y)): 0.0665 - KL Divergence: 5.6263
Epoch 4/200
782/782 [=====] - 55s 71ms/step - d_loss: 0.1900 - g_loss: 3.5944 -
D(x|y): 0.4976 - D(G(z|y)): 0.0606 - KL Divergence: 6.1849
Epoch 5/200
782/782 [=====] - 55s 71ms/step - d_loss: 0.2508 - g_loss: 2.9719 -
D(x|y): 0.4979 - D(G(z|y)): 0.0801 - KL Divergence: 5.6865
1/1 [=====] - 0s 28ms/step



Generator Checkpoint - New cGAN/generator-epoch-5.h5

Epoch 6/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2900 - g_loss: 2.8784 - D(x|y): 0.5025 - D(G(z|y)): 0.0886 - KL Divergence: 4.3030

Epoch 7/200

782/782 [=====] - 55s 70ms/step - d_loss: 0.2638 - g_loss: 2.9703 - D(x|y): 0.5016 - D(G(z|y)): 0.0794 - KL Divergence: 4.8253

Epoch 8/200

782/782 [=====] - 54s 70ms/step - d_loss: 0.3211 - g_loss: 2.6753 - D(x|y): 0.5003 - D(G(z|y)): 0.1039 - KL Divergence: 4.9567

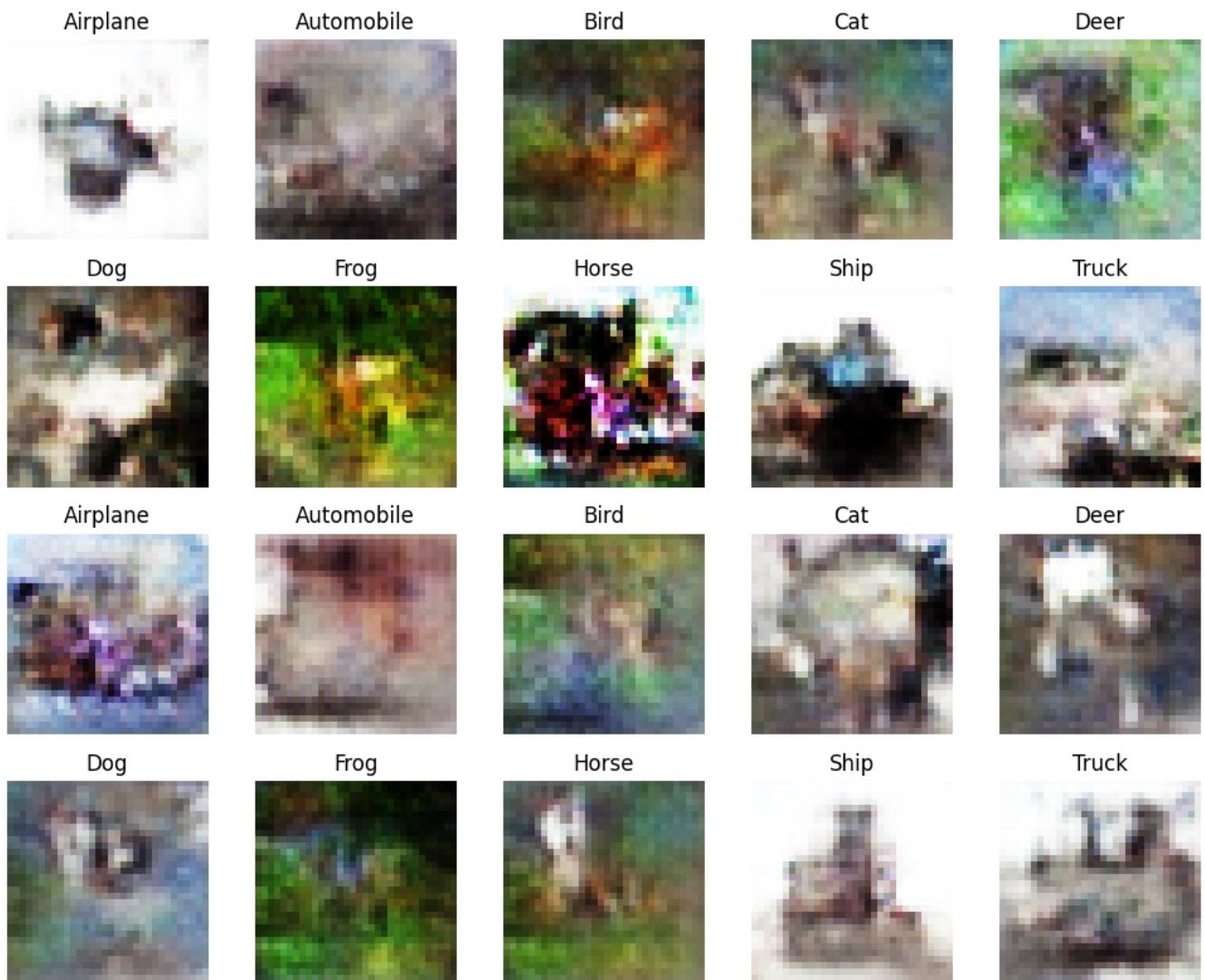
Epoch 9/200

782/782 [=====] - 53s 68ms/step - d_loss: 0.2964 - g_loss: 2.8504 - D(x|y): 0.5006 - D(G(z|y)): 0.1076 - KL Divergence: 4.7245

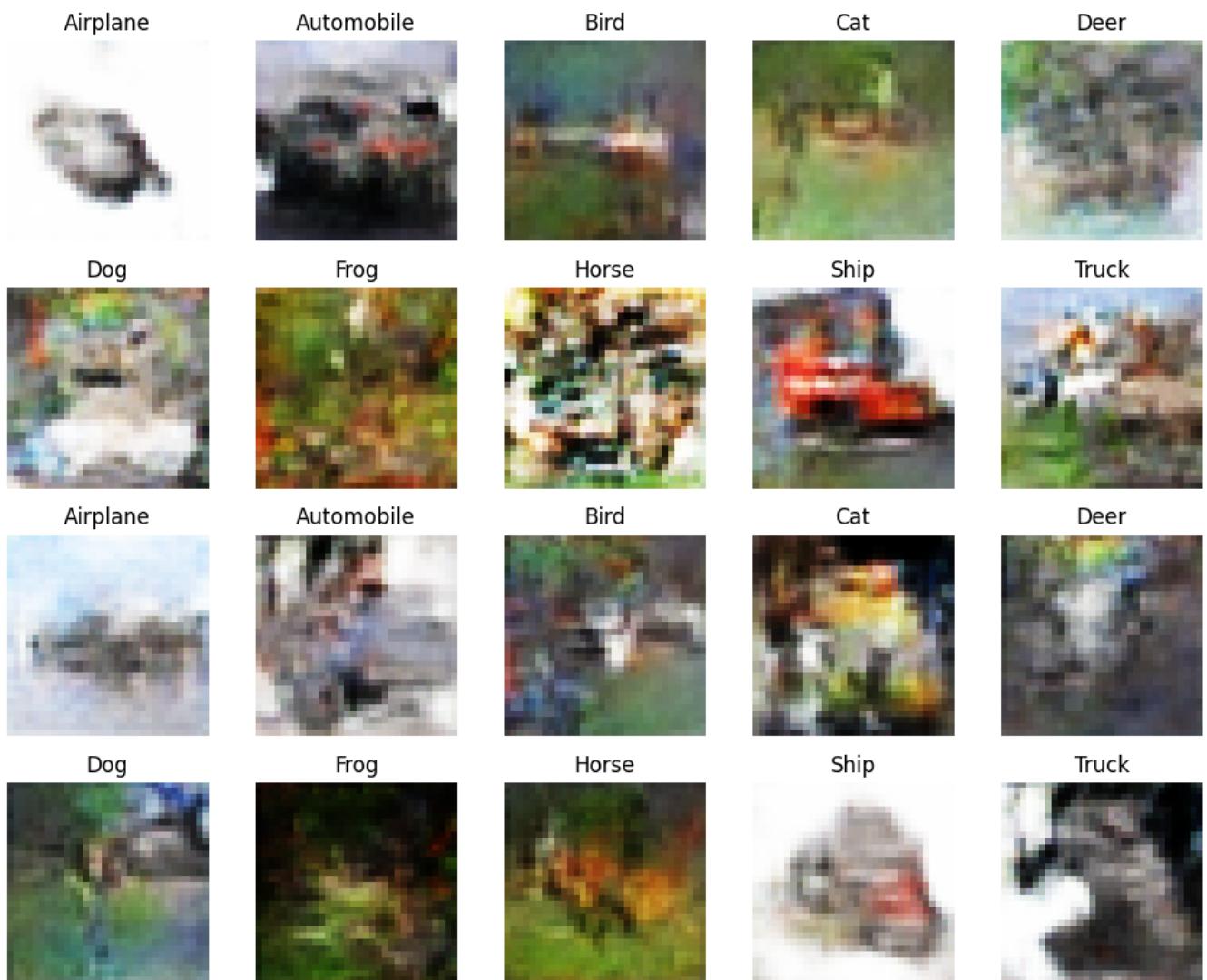
Epoch 10/200

782/782 [=====] - 53s 68ms/step - d_loss: 0.2979 - g_loss: 2.7473 - D(x|y): 0.4999 - D(G(z|y)): 0.1211 - KL Divergence: 4.7051

1/1 [=====] - 0s 33ms/step



Generator Checkpoint - New cGAN/generator-epoch-10.h5
Epoch 11/200
782/782 [=====] - 54s 69ms/step - d_loss: 0.3139 - g_loss: 2.6440 -
D(x|y): 0.4996 - D(G(z|y)): 0.1330 - KL Divergence: 4.6347
Epoch 12/200
782/782 [=====] - 54s 70ms/step - d_loss: 0.2899 - g_loss: 2.7278 -
D(x|y): 0.5001 - D(G(z|y)): 0.1310 - KL Divergence: 4.7942
Epoch 13/200
782/782 [=====] - 54s 69ms/step - d_loss: 0.2968 - g_loss: 2.6324 -
D(x|y): 0.5011 - D(G(z|y)): 0.1422 - KL Divergence: 4.5704
Epoch 14/200
782/782 [=====] - 54s 69ms/step - d_loss: 0.2985 - g_loss: 2.6300 -
D(x|y): 0.5006 - D(G(z|y)): 0.1444 - KL Divergence: 4.6995
Epoch 15/200
782/782 [=====] - 54s 69ms/step - d_loss: 0.3298 - g_loss: 2.5145 -
D(x|y): 0.4994 - D(G(z|y)): 0.1618 - KL Divergence: 4.6561
1/1 [=====] - 0s 28ms/step



Generator Checkpoint - New cGAN/generator-epoch-15.h5

Epoch 16/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.3063 - g_loss: 2.5317 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.1586 - KL Divergence: 4.6985

Epoch 17/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.3093 - g_loss: 2.5373 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.1588 - KL Divergence: 4.6638

Epoch 18/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2971 - g_loss: 2.6530 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.1519 - KL Divergence: 4.7238

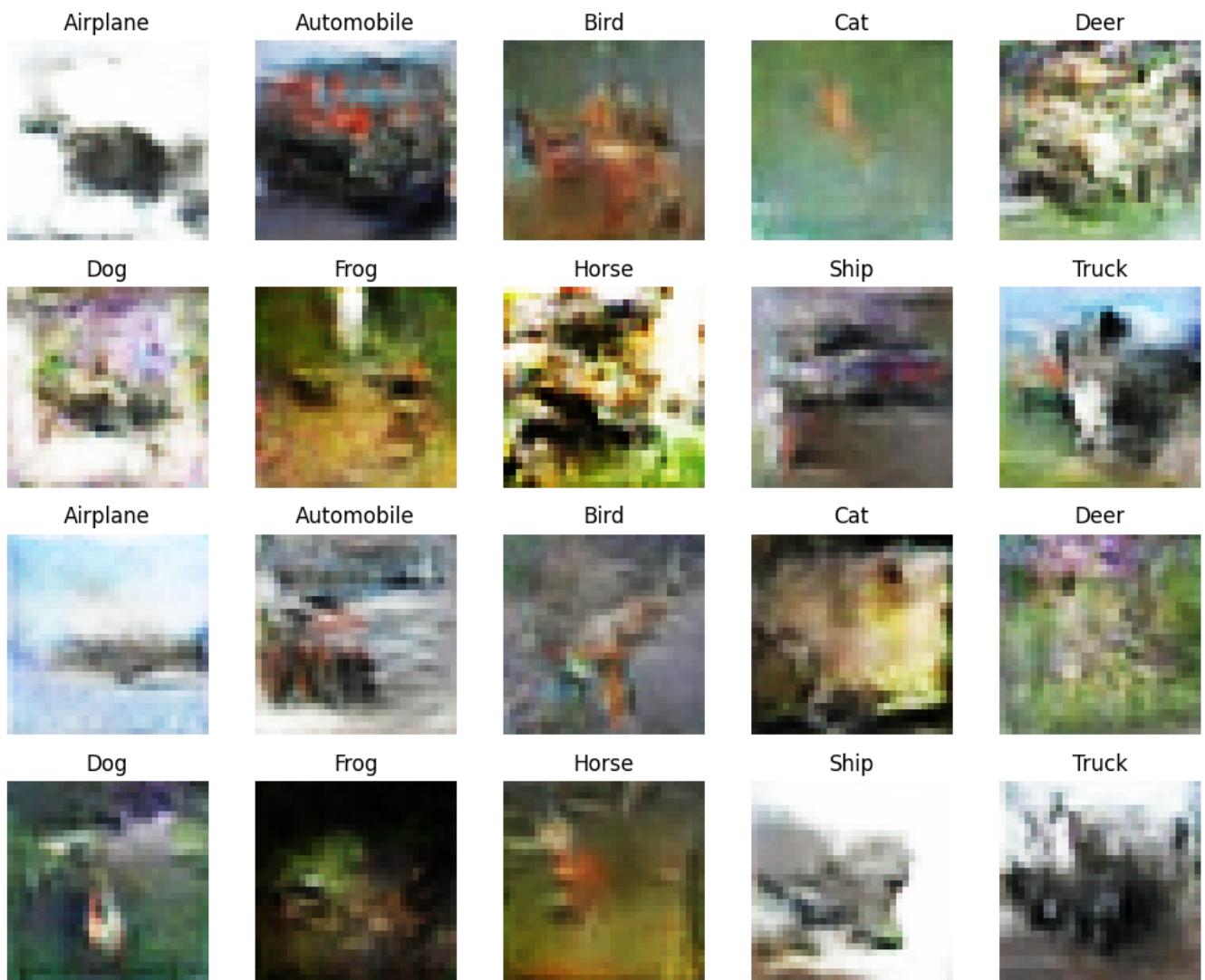
Epoch 19/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2996 - g_loss: 2.5974 -
 $D(x|y)$: 0.5007 - $D(G(z|y))$: 0.1553 - KL Divergence: 4.7140

Epoch 20/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2839 - g_loss: 2.6496 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.1508 - KL Divergence: 4.7201

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - New cGAN/generator-epoch-20.h5

Epoch 21/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2788 - g_loss: 2.6916 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.1483 - KL Divergence: 4.8230

Epoch 22/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2773 - g_loss: 2.7279 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.1480 - KL Divergence: 4.7877

Epoch 23/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2898 - g_loss: 2.6814 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.1564 - KL Divergence: 4.7436

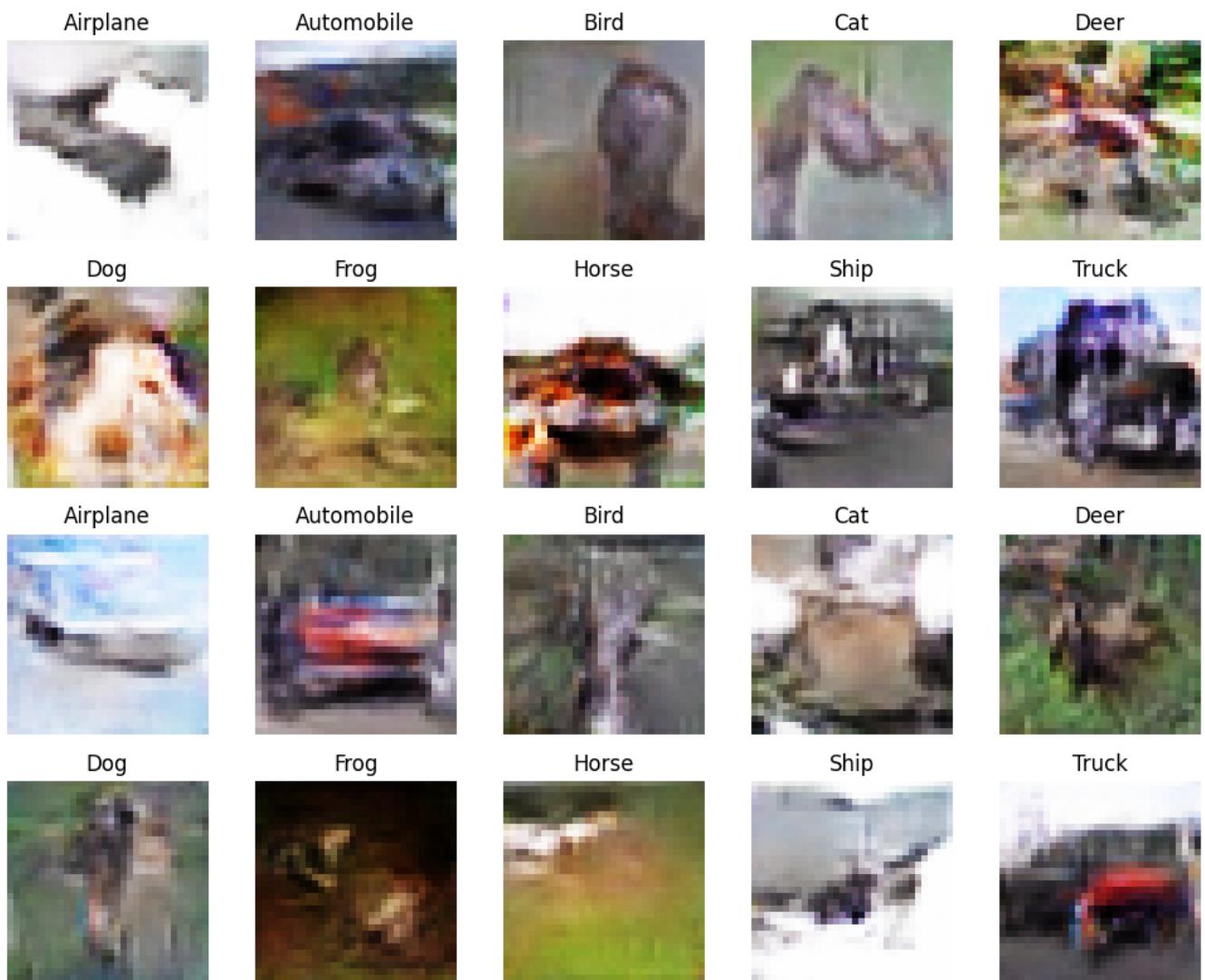
Epoch 24/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2827 - g_loss: 2.6113 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.1602 - KL Divergence: 4.7403

Epoch 25/200

782/782 [=====] - 54s 69ms/step - d_loss: 0.2849 - g_loss: 2.6963 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.1573 - KL Divergence: 4.7825

1/1 [=====] - 0s 30ms/step



Generator Checkpoint - New cGAN/generator-epoch-25.h5

Epoch 26/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.2720 - g_loss: 2.8045 -
 $D(x|y): 0.4997$ - $D(G(z|y)): 0.1498$ - KL Divergence: 4.8034

Epoch 27/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.2653 - g_loss: 2.8807 -
 $D(x|y): 0.5000$ - $D(G(z|y)): 0.1462$ - KL Divergence: 4.6746

Epoch 28/200

782/782 [=====] - 82s 104ms/step - d_loss: 0.2570 - g_loss: 2.9827 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.1418$ - KL Divergence: 4.6800

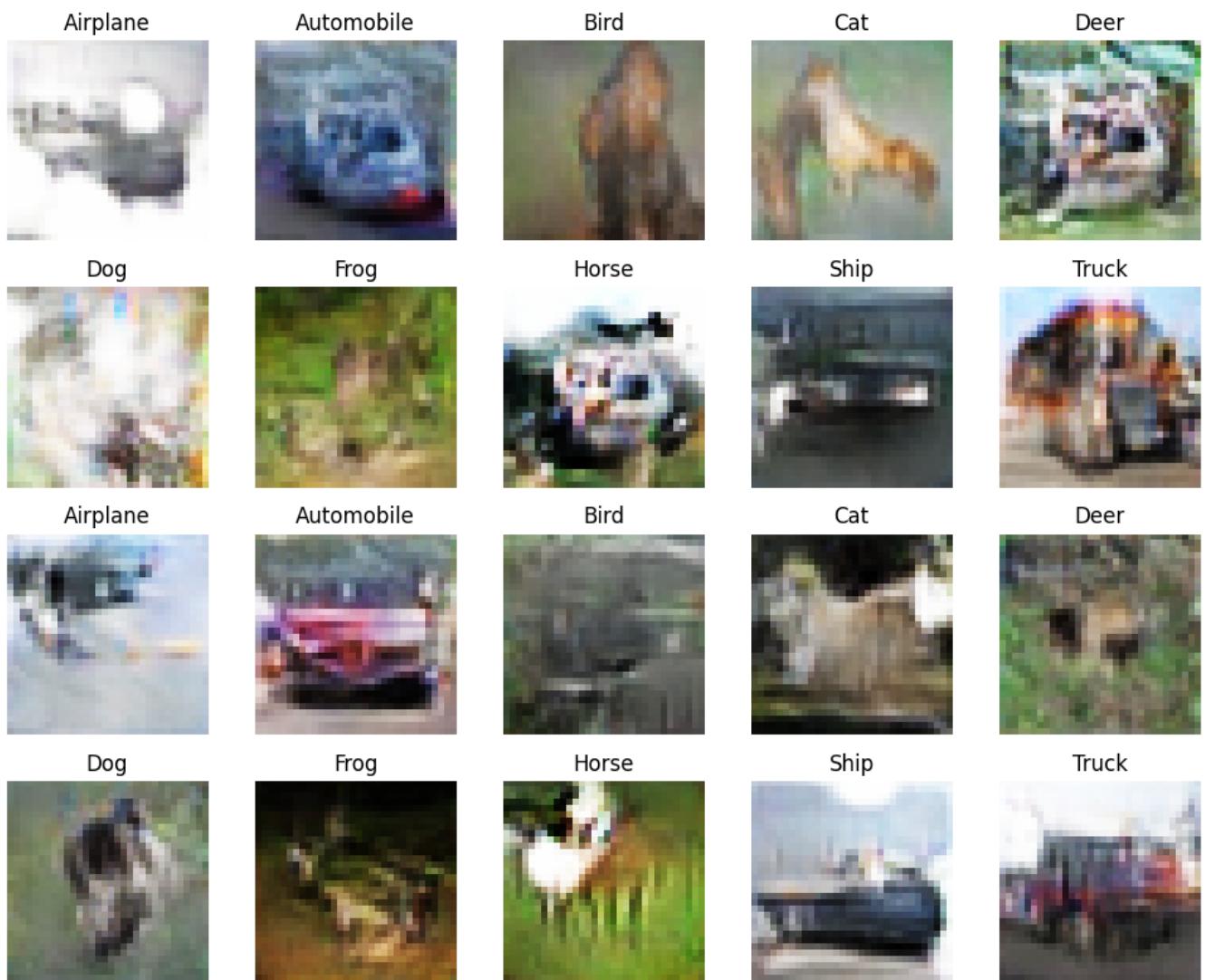
Epoch 29/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.2506 - g_loss: 3.0585 -
 $D(x|y): 0.4998$ - $D(G(z|y)): 0.1376$ - KL Divergence: 4.7081

Epoch 30/200

782/782 [=====] - 76s 98ms/step - d_loss: 0.2441 - g_loss: 3.1226 -
 $D(x|y): 0.5007$ - $D(G(z|y)): 0.1365$ - KL Divergence: 4.7073

1/1 [=====] - 0s 23ms/step



Generator Checkpoint - New cGAN/generator-epoch-30.h5

Epoch 31/200

782/782 [=====] - 80s 103ms/step - d_loss: 0.2550 - g_loss: 3.0789 -
 $D(x|y)$: 0.5007 - $D(G(z|y))$: 0.1429 - KL Divergence: 4.7381

Epoch 32/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.2830 - g_loss: 3.0047 -
 $D(x|y)$: 0.5008 - $D(G(z|y))$: 0.1509 - KL Divergence: 4.7244

Epoch 33/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.2281 - g_loss: 3.3576 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.1266 - KL Divergence: 4.7971

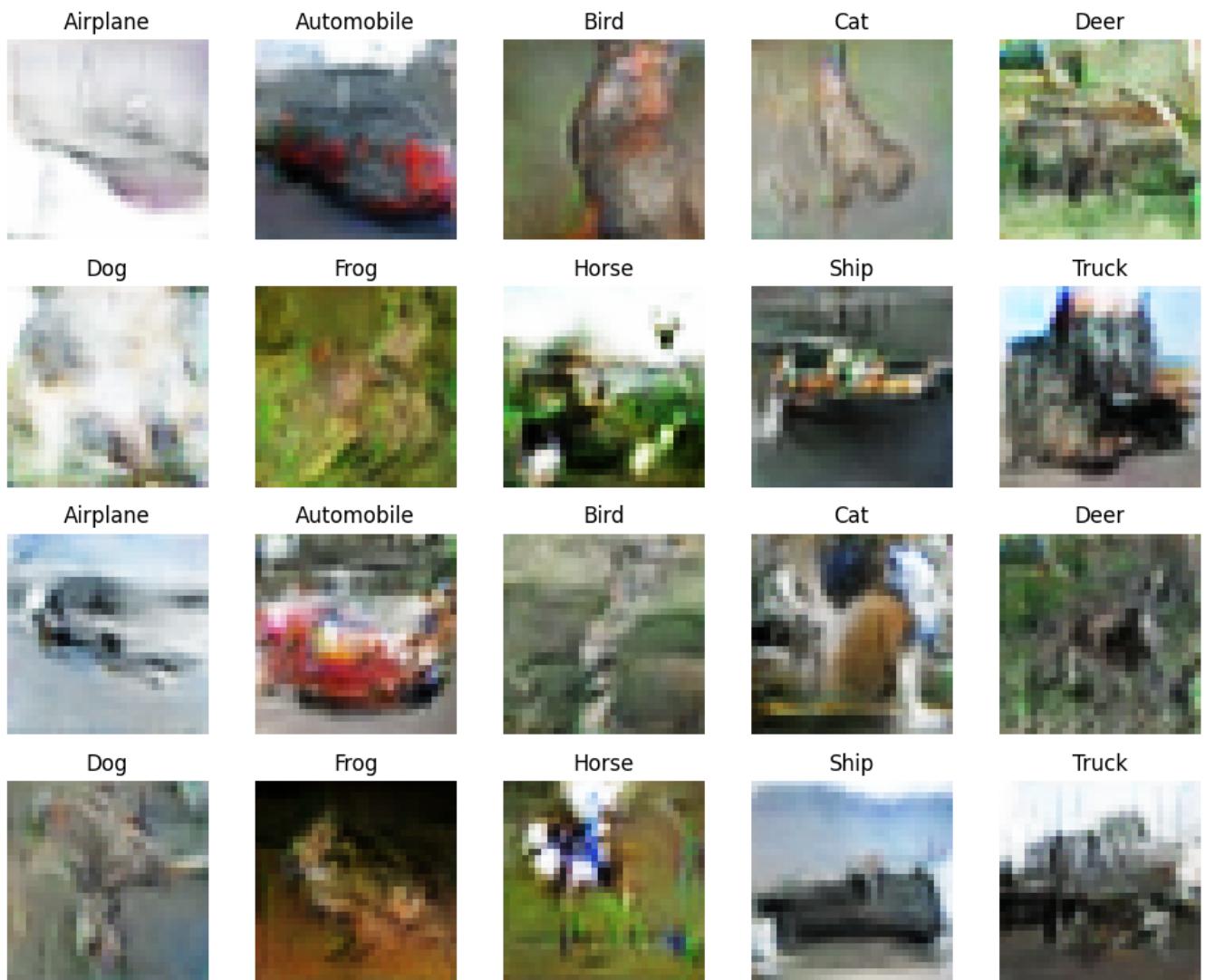
Epoch 34/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.2190 - g_loss: 3.4356 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.1221 - KL Divergence: 4.6932

Epoch 35/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.2183 - g_loss: 3.5289 -
 $D(x|y)$: 0.4996 - $D(G(z|y))$: 0.1205 - KL Divergence: 4.7403

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New cGAN/generator-epoch-35.h5

Epoch 36/200

782/782 [=====] - 77s 98ms/step - d_loss: 0.2157 - g_loss: 3.5700 -
 $D(x|y): 0.5006$ - $D(G(z|y)): 0.1187$ - KL Divergence: 4.7593

Epoch 37/200

782/782 [=====] - 80s 102ms/step - d_loss: 0.2169 - g_loss: 3.6181 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.1210$ - KL Divergence: 4.6791

Epoch 38/200

782/782 [=====] - 80s 102ms/step - d_loss: 0.1977 - g_loss: 3.7868 -
 $D(x|y): 0.5006$ - $D(G(z|y)): 0.1106$ - KL Divergence: 4.7311

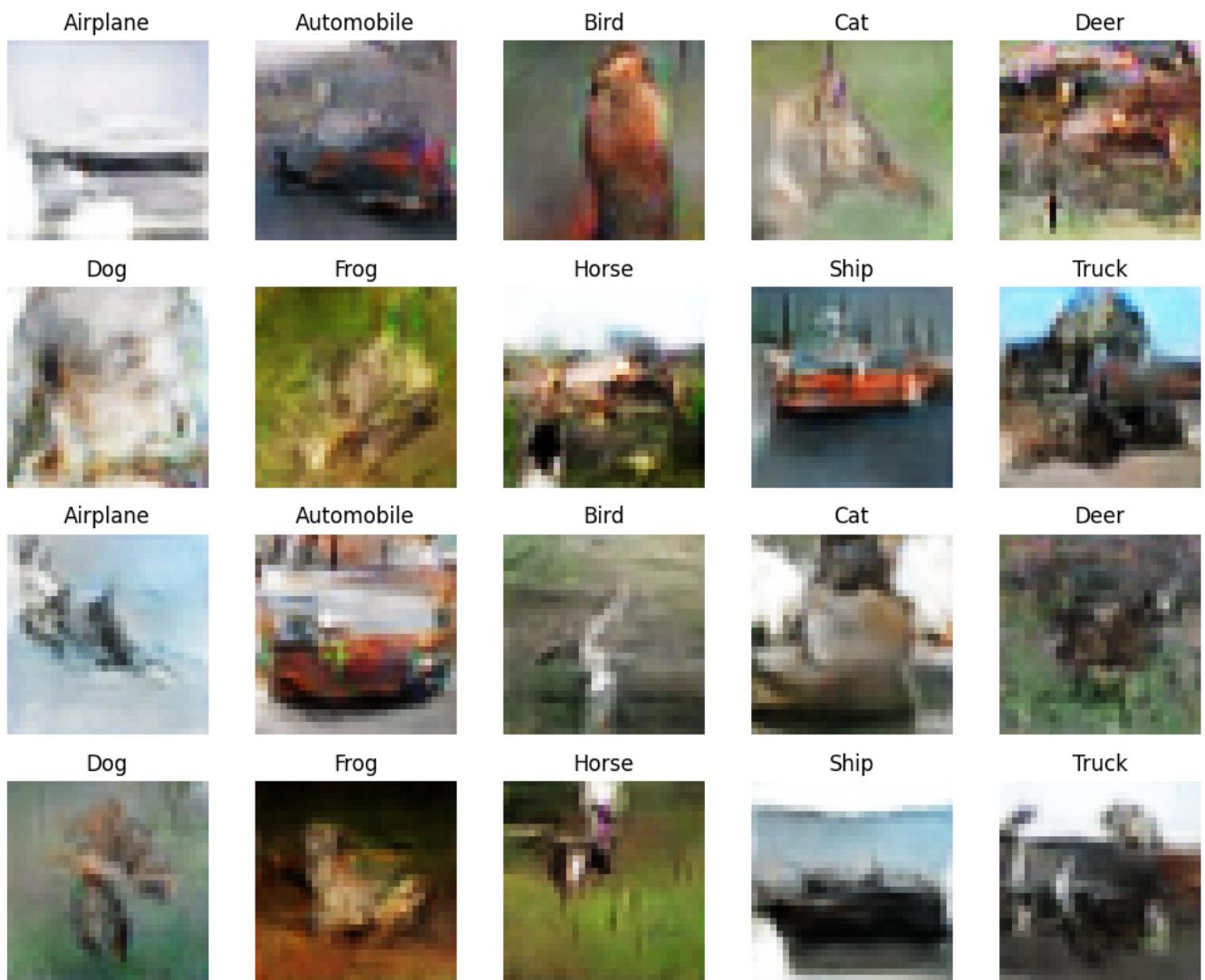
Epoch 39/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.1946 - g_loss: 3.8984 -
 $D(x|y): 0.4999$ - $D(G(z|y)): 0.1069$ - KL Divergence: 4.7991

Epoch 40/200

782/782 [=====] - 80s 102ms/step - d_loss: 0.1892 - g_loss: 3.9645 -
 $D(x|y): 0.5007$ - $D(G(z|y)): 0.1052$ - KL Divergence: 4.6082

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New cGAN/generator-epoch-40.h5

Epoch 41/200

782/782 [=====] - 80s 102ms/step - d_loss: 0.1858 - g_loss: 4.0267 - D(x|y): 0.5007 - D(G(z|y)): 0.1040 - KL Divergence: 4.8212

Epoch 42/200

782/782 [=====] - 80s 103ms/step - d_loss: 0.1892 - g_loss: 4.0255 - D(x|y): 0.4991 - D(G(z|y)): 0.1025 - KL Divergence: 4.7064

Epoch 43/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.1742 - g_loss: 4.1019 - D(x|y): 0.5000 - D(G(z|y)): 0.1003 - KL Divergence: 4.7073

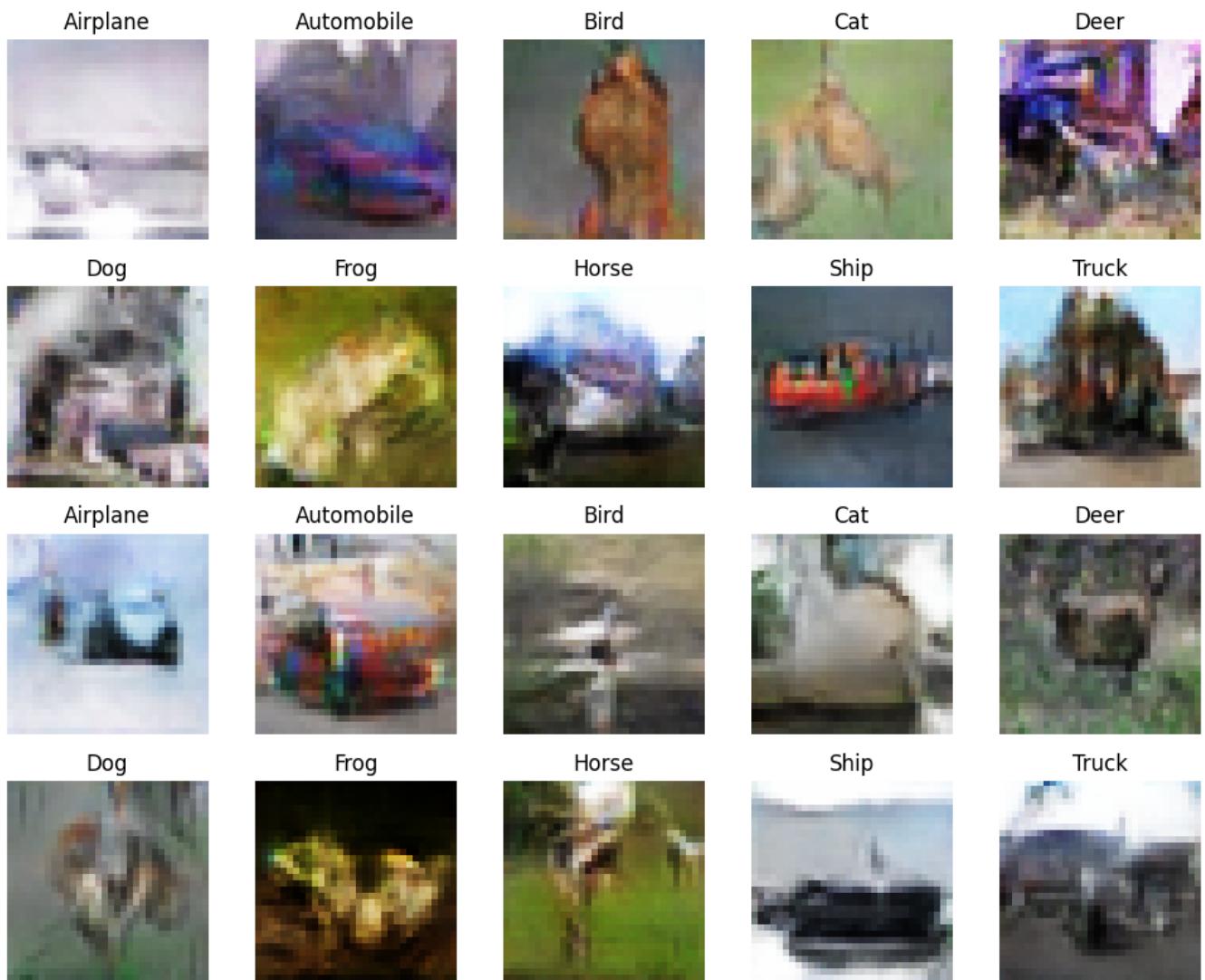
Epoch 44/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.1789 - g_loss: 4.2403 - D(x|y): 0.5000 - D(G(z|y)): 0.0995 - KL Divergence: 4.6498

Epoch 45/200

782/782 [=====] - 80s 103ms/step - d_loss: 0.1643 - g_loss: 4.4078 - D(x|y): 0.4999 - D(G(z|y)): 0.0943 - KL Divergence: 4.6208

1/1 [=====] - 0s 22ms/step



Generator Checkpoint - New cGAN/generator-epoch-45.h5

Epoch 46/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.1672 - g_loss: 4.4206 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0940 - KL Divergence: 4.7090

Epoch 47/200

782/782 [=====] - 61s 79ms/step - d_loss: 0.1655 - g_loss: 4.4484 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0921 - KL Divergence: 4.6732

Epoch 48/200

782/782 [=====] - 57s 73ms/step - d_loss: 0.1604 - g_loss: 4.6099 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0886 - KL Divergence: 4.5495

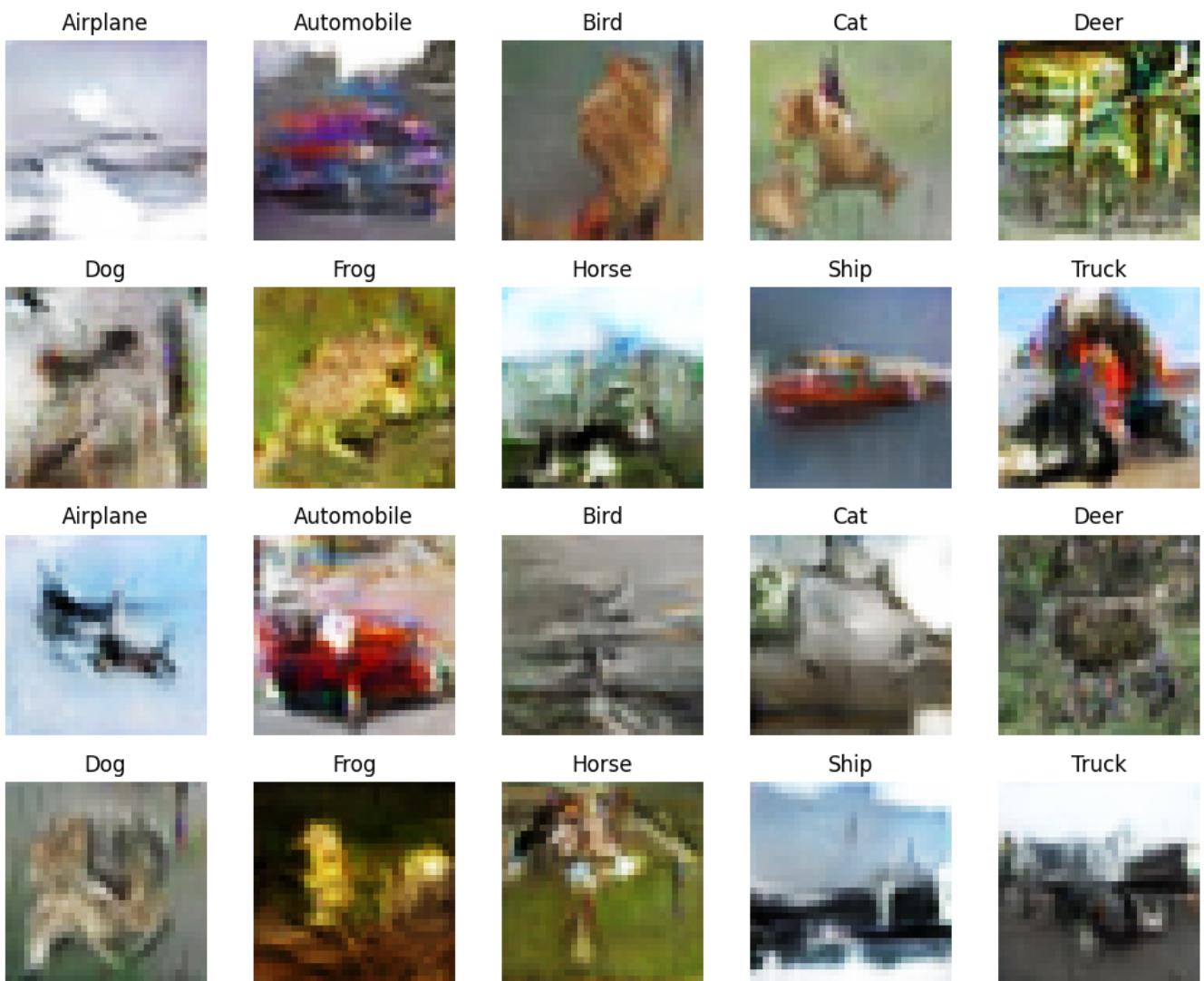
Epoch 49/200

782/782 [=====] - 61s 78ms/step - d_loss: 0.1574 - g_loss: 4.6186 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0880 - KL Divergence: 4.6346

Epoch 50/200

782/782 [=====] - 51s 65ms/step - d_loss: 0.1554 - g_loss: 4.6344 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0880 - KL Divergence: 4.6760

1/1 [=====] - 0s 30ms/step



Generator Checkpoint - New cGAN/generator-epoch-50.h5

Epoch 51/200

```
782/782 [=====] - 59s 75ms/step - d_loss: 0.1605 - g_loss: 4.6824 -
D(x|y): 0.5002 - D(G(z|y)): 0.0883 - KL Divergence: 4.4981
```

Epoch 52/200

```
782/782 [=====] - 60s 77ms/step - d_loss: 0.1482 - g_loss: 4.8821 -
D(x|y): 0.4997 - D(G(z|y)): 0.0813 - KL Divergence: 4.3949
```

Epoch 53/200

```
782/782 [=====] - 57s 72ms/step - d_loss: 0.1527 - g_loss: 4.8996 -
D(x|y): 0.5007 - D(G(z|y)): 0.0854 - KL Divergence: 4.5783
```

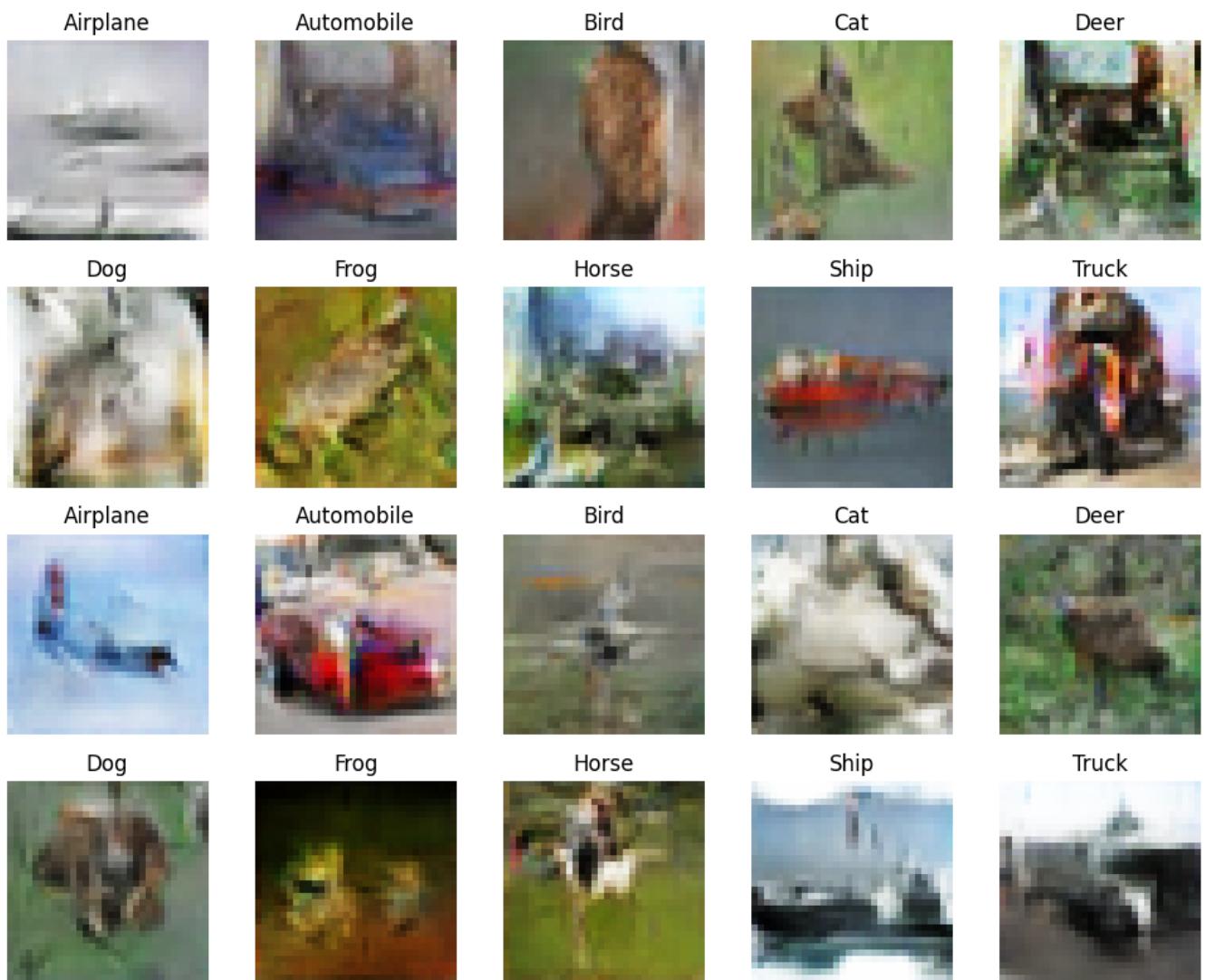
Epoch 54/200

```
782/782 [=====] - 54s 70ms/step - d_loss: 0.1501 - g_loss: 4.9697 -
D(x|y): 0.5000 - D(G(z|y)): 0.0812 - KL Divergence: 4.4994
```

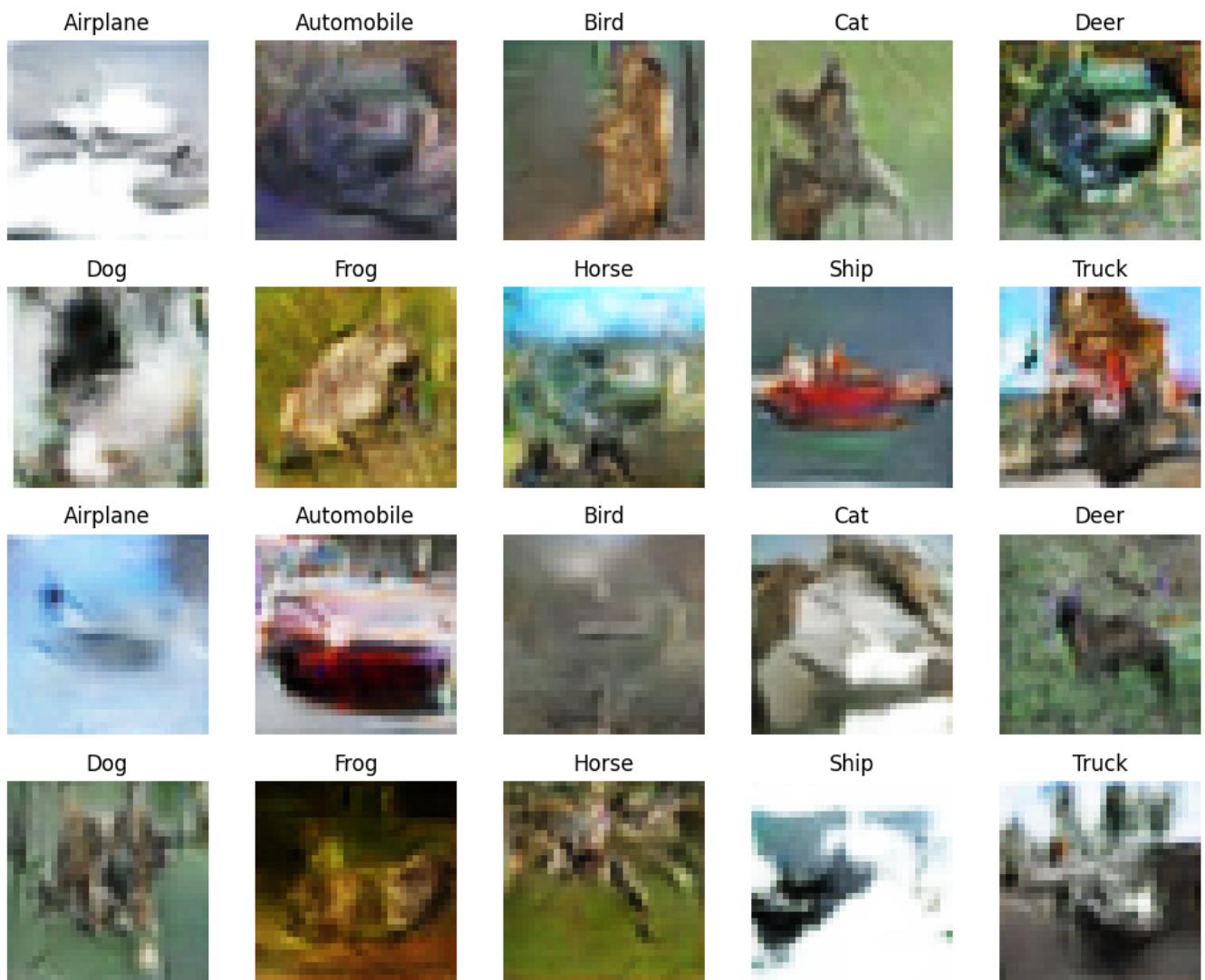
Epoch 55/200

```
782/782 [=====] - 54s 69ms/step - d_loss: 0.1480 - g_loss: 4.9598 -
D(x|y): 0.5003 - D(G(z|y)): 0.0812 - KL Divergence: 4.5852
```

1/1 [=====] - 0s 30ms/step



Generator Checkpoint - New cGAN/generator-epoch-55.h5
 Epoch 56/200
 782/782 [=====] - 54s 70ms/step - d_loss: 0.1427 - g_loss: 5.0395 -
 $D(x|y): 0.5009$ - $D(G(z|y)): 0.0771$ - KL Divergence: 4.6244
 Epoch 57/200
 782/782 [=====] - 55s 70ms/step - d_loss: 0.1407 - g_loss: 5.1246 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.0794$ - KL Divergence: 4.6944
 Epoch 58/200
 782/782 [=====] - 56s 71ms/step - d_loss: 0.3012 - g_loss: 4.1907 -
 $D(x|y): 0.5017$ - $D(G(z|y)): 0.1332$ - KL Divergence: 4.6172
 Epoch 59/200
 782/782 [=====] - 49s 63ms/step - d_loss: 0.1363 - g_loss: 4.9949 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0756$ - KL Divergence: 4.5504
 Epoch 60/200
 782/782 [=====] - 55s 71ms/step - d_loss: 0.1384 - g_loss: 5.1037 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0769$ - KL Divergence: 4.6315
 1/1 [=====] - 0s 47ms/step



Generator Checkpoint - New cGAN/generator-epoch-60.h5

Epoch 61/200

782/782 [=====] - 59s 76ms/step - d_loss: 0.1499 - g_loss: 5.1048 -
 $D(x|y)$: 0.4997 - $D(G(z|y))$: 0.0788 - KL Divergence: 4.6915

Epoch 62/200

782/782 [=====] - 62s 79ms/step - d_loss: 0.1363 - g_loss: 5.2132 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0758 - KL Divergence: 4.6619

Epoch 63/200

782/782 [=====] - 61s 78ms/step - d_loss: 0.1366 - g_loss: 5.2235 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0768 - KL Divergence: 4.7913

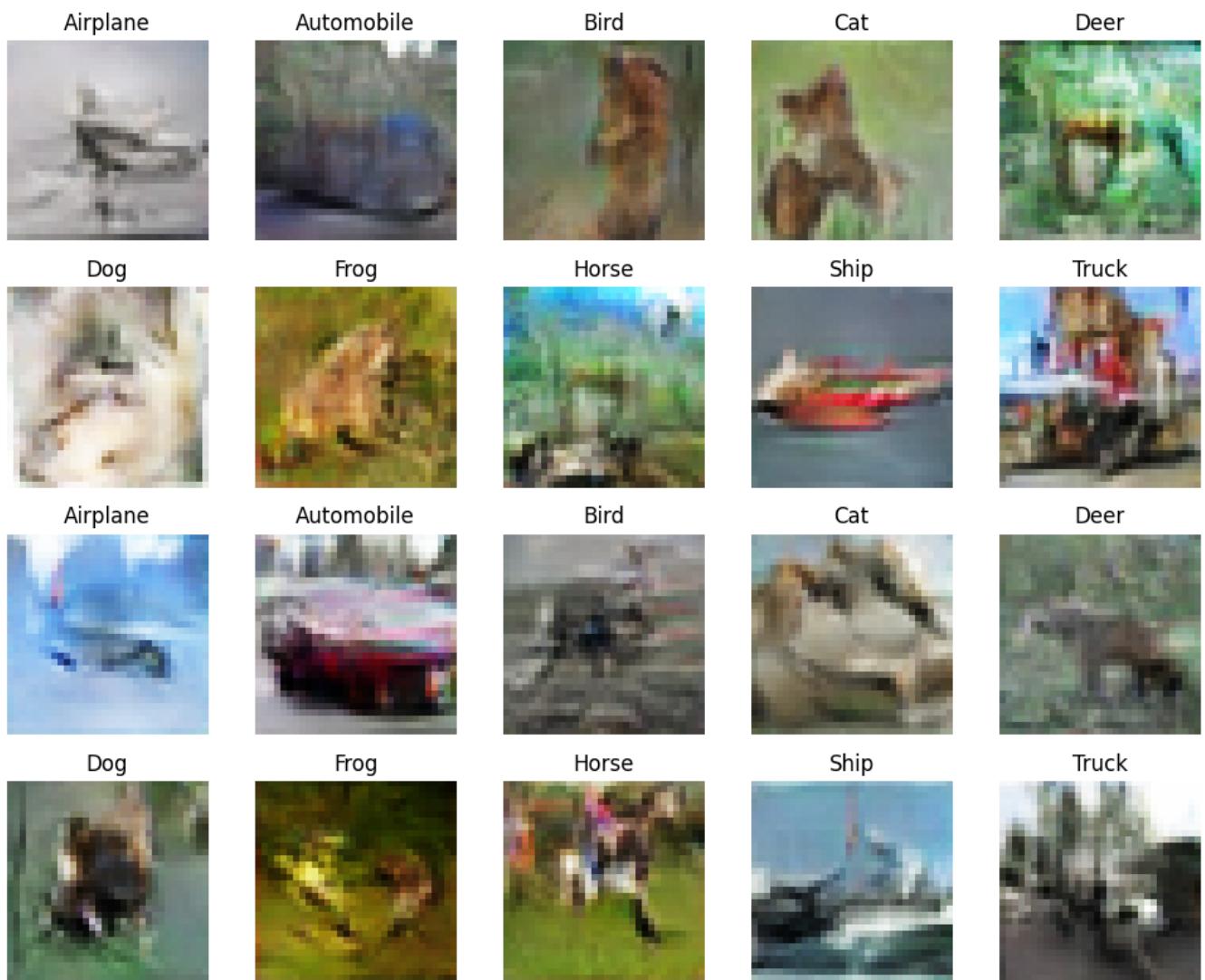
Epoch 64/200

782/782 [=====] - 55s 70ms/step - d_loss: 0.1470 - g_loss: 5.2257 -
 $D(x|y)$: 0.4997 - $D(G(z|y))$: 0.0783 - KL Divergence: 4.6809

Epoch 65/200

782/782 [=====] - 51s 65ms/step - d_loss: 0.1313 - g_loss: 5.3987 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0731 - KL Divergence: 4.5989

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New cGAN/generator-epoch-65.h5

Epoch 66/200

782/782 [=====] - 51s 65ms/step - d_loss: 0.1314 - g_loss: 5.4156 - D(x|y): 0.5002 - D(G(z|y)): 0.0730 - KL Divergence: 4.6488

Epoch 67/200

782/782 [=====] - 55s 70ms/step - d_loss: 0.1276 - g_loss: 5.4792 - D(x|y): 0.5002 - D(G(z|y)): 0.0712 - KL Divergence: 4.5688

Epoch 68/200

782/782 [=====] - 59s 76ms/step - d_loss: 0.1357 - g_loss: 5.4498 - D(x|y): 0.5007 - D(G(z|y)): 0.0710 - KL Divergence: 4.7601

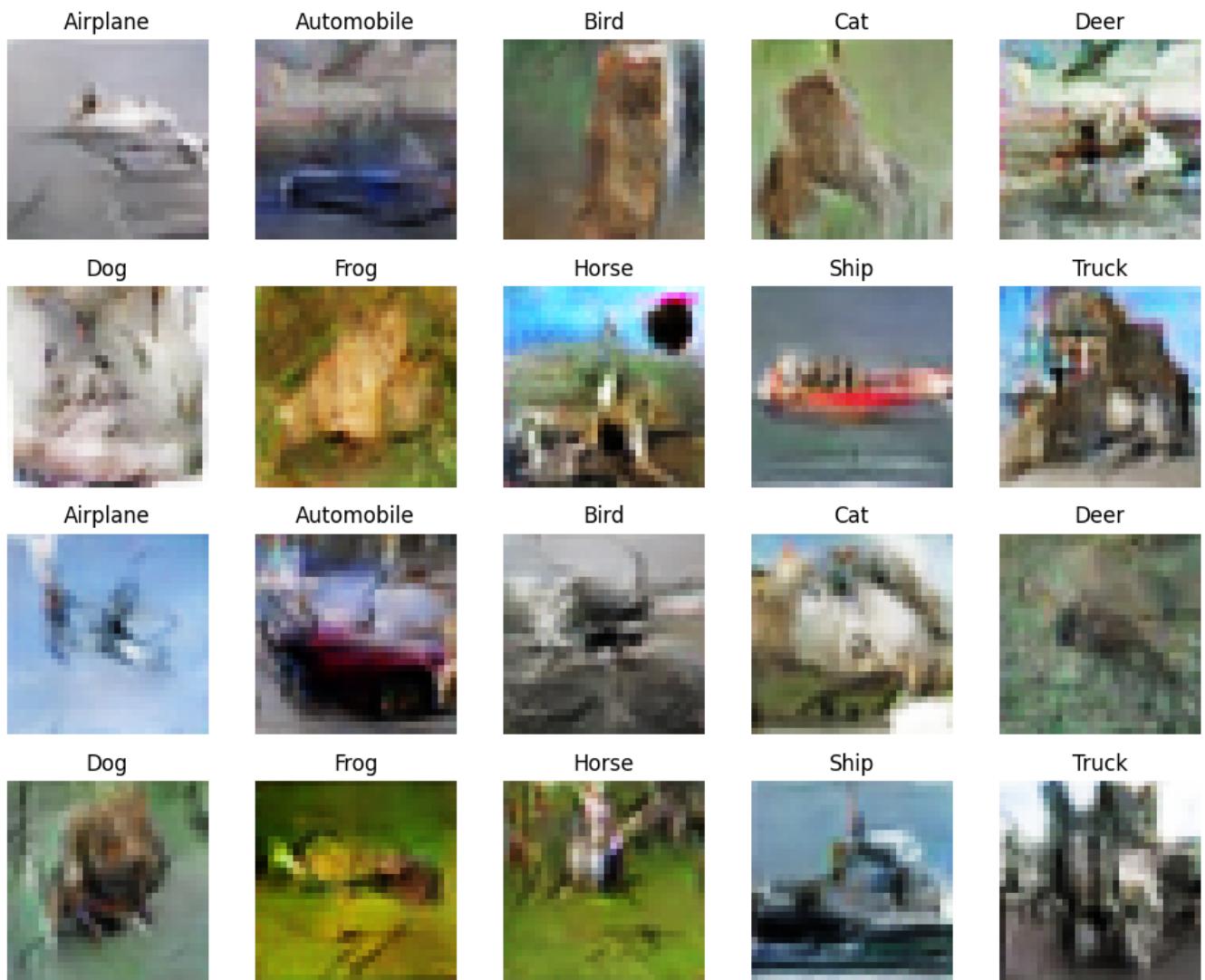
Epoch 69/200

782/782 [=====] - 57s 73ms/step - d_loss: 0.1250 - g_loss: 5.5100 - D(x|y): 0.4999 - D(G(z|y)): 0.0698 - KL Divergence: 4.8093

Epoch 70/200

782/782 [=====] - 57s 73ms/step - d_loss: 0.1275 - g_loss: 5.5687 - D(x|y): 0.5005 - D(G(z|y)): 0.0696 - KL Divergence: 4.5884

1/1 [=====] - 0s 39ms/step



Generator Checkpoint - New cGAN/generator-epoch-70.h5

Epoch 71/200

782/782 [=====] - 57s 72ms/step - d_loss: 0.1274 - g_loss: 5.6669 - D(x|y): 0.4999 - D(G(z|y)): 0.0688 - KL Divergence: 4.6138

Epoch 72/200

782/782 [=====] - 62s 79ms/step - d_loss: 0.1258 - g_loss: 5.7240 - D(x|y): 0.5004 - D(G(z|y)): 0.0683 - KL Divergence: 4.6379

Epoch 73/200

782/782 [=====] - 59s 76ms/step - d_loss: 0.1209 - g_loss: 5.8340 - D(x|y): 0.5000 - D(G(z|y)): 0.0657 - KL Divergence: 4.6185

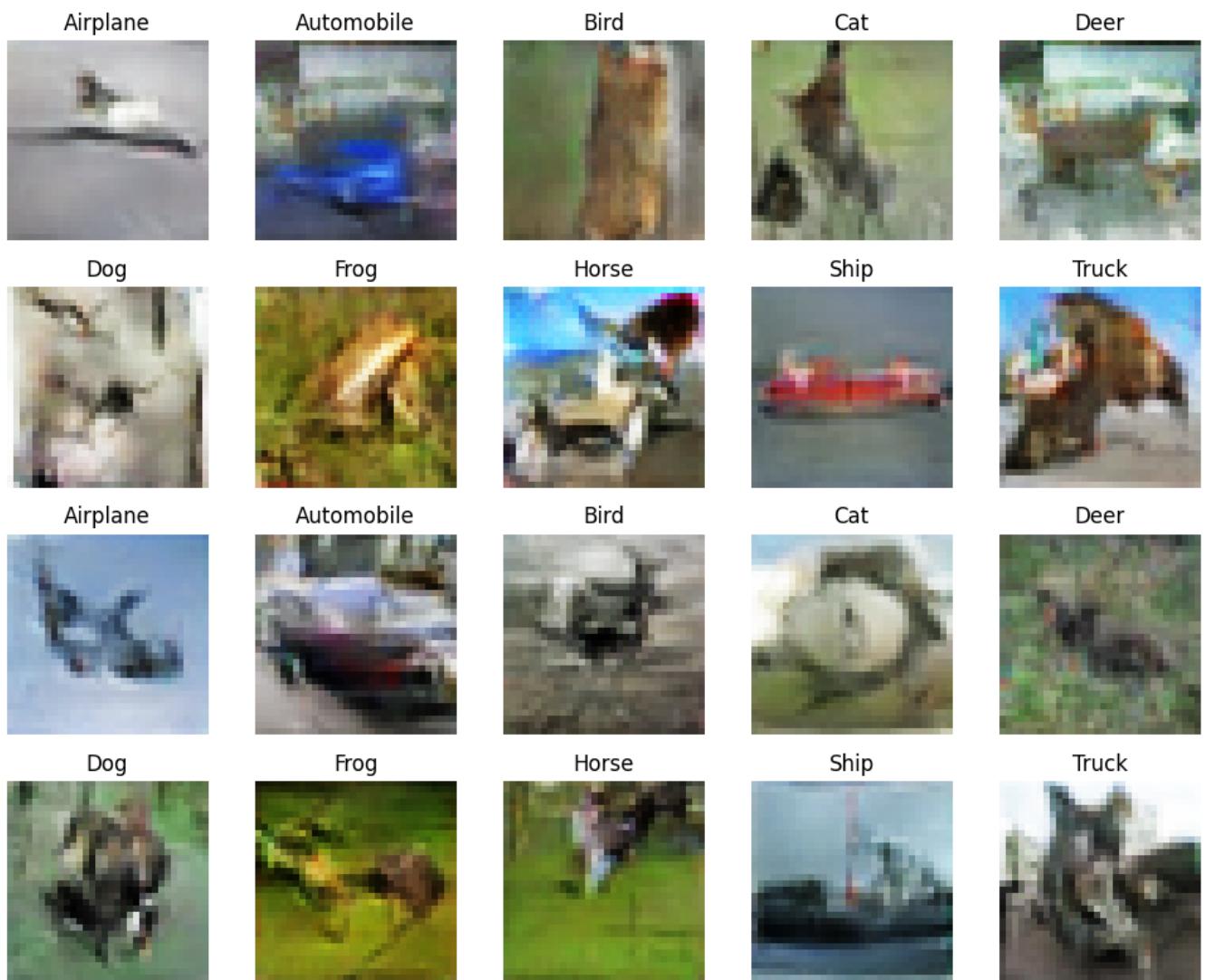
Epoch 74/200

782/782 [=====] - 58s 74ms/step - d_loss: 0.1214 - g_loss: 5.8127 - D(x|y): 0.5004 - D(G(z|y)): 0.0649 - KL Divergence: 4.5770

Epoch 75/200

782/782 [=====] - 64s 82ms/step - d_loss: 0.1245 - g_loss: 5.8535 - D(x|y): 0.5005 - D(G(z|y)): 0.0651 - KL Divergence: 4.7295

1/1 [=====] - 0s 77ms/step



Generator Checkpoint - New cGAN/generator-epoch-75.h5

Epoch 76/200

782/782 [=====] - 59s 76ms/step - d_loss: 0.1164 - g_loss: 5.9926 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0653$ - KL Divergence: 4.6654

Epoch 77/200

782/782 [=====] - 58s 75ms/step - d_loss: 0.1216 - g_loss: 5.9395 -
 $D(x|y): 0.4997$ - $D(G(z|y)): 0.0665$ - KL Divergence: 4.6298

Epoch 78/200

782/782 [=====] - 56s 72ms/step - d_loss: 0.1142 - g_loss: 6.0926 -
 $D(x|y): 0.5001$ - $D(G(z|y)): 0.0598$ - KL Divergence: 4.6145

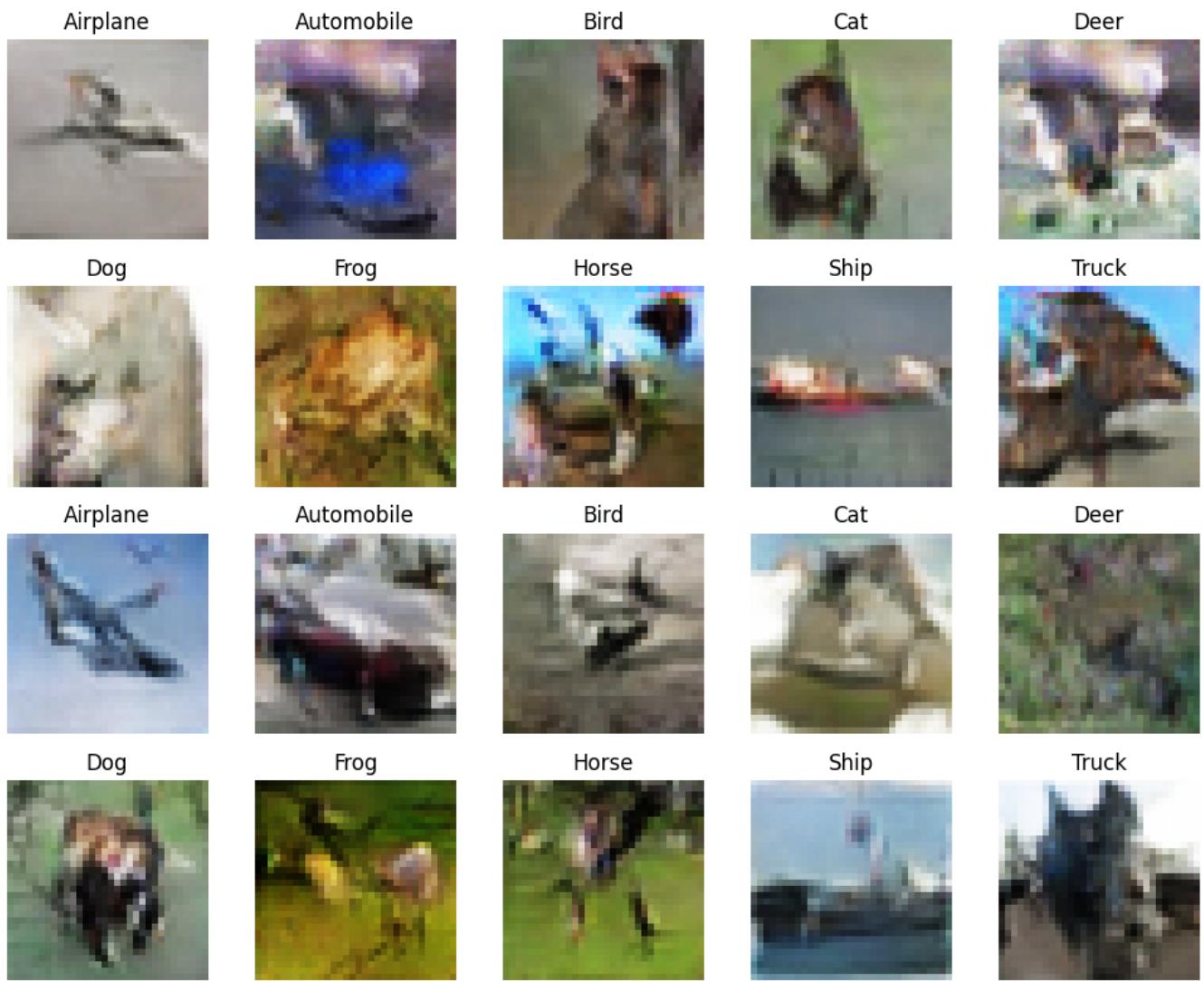
Epoch 79/200

782/782 [=====] - 56s 71ms/step - d_loss: 0.1221 - g_loss: 5.9829 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0650$ - KL Divergence: 4.6439

Epoch 80/200

782/782 [=====] - 56s 72ms/step - d_loss: 0.1159 - g_loss: 6.0471 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0630$ - KL Divergence: 4.6721

1/1 [=====] - 0s 37ms/step



Generator Checkpoint - New cGAN/generator-epoch-80.h5

Epoch 81/200

782/782 [=====] - 56s 72ms/step - d_loss: 0.1146 - g_loss: 6.1334 - D(x|y): 0.5003 - D(G(z|y)): 0.0597 - KL Divergence: 4.4338

Epoch 82/200

782/782 [=====] - 56s 72ms/step - d_loss: 0.1171 - g_loss: 6.0960 - D(x|y): 0.4996 - D(G(z|y)): 0.0635 - KL Divergence: 4.6909

Epoch 83/200

782/782 [=====] - 56s 72ms/step - d_loss: 0.1115 - g_loss: 6.2260 - D(x|y): 0.5000 - D(G(z|y)): 0.0613 - KL Divergence: 4.6485

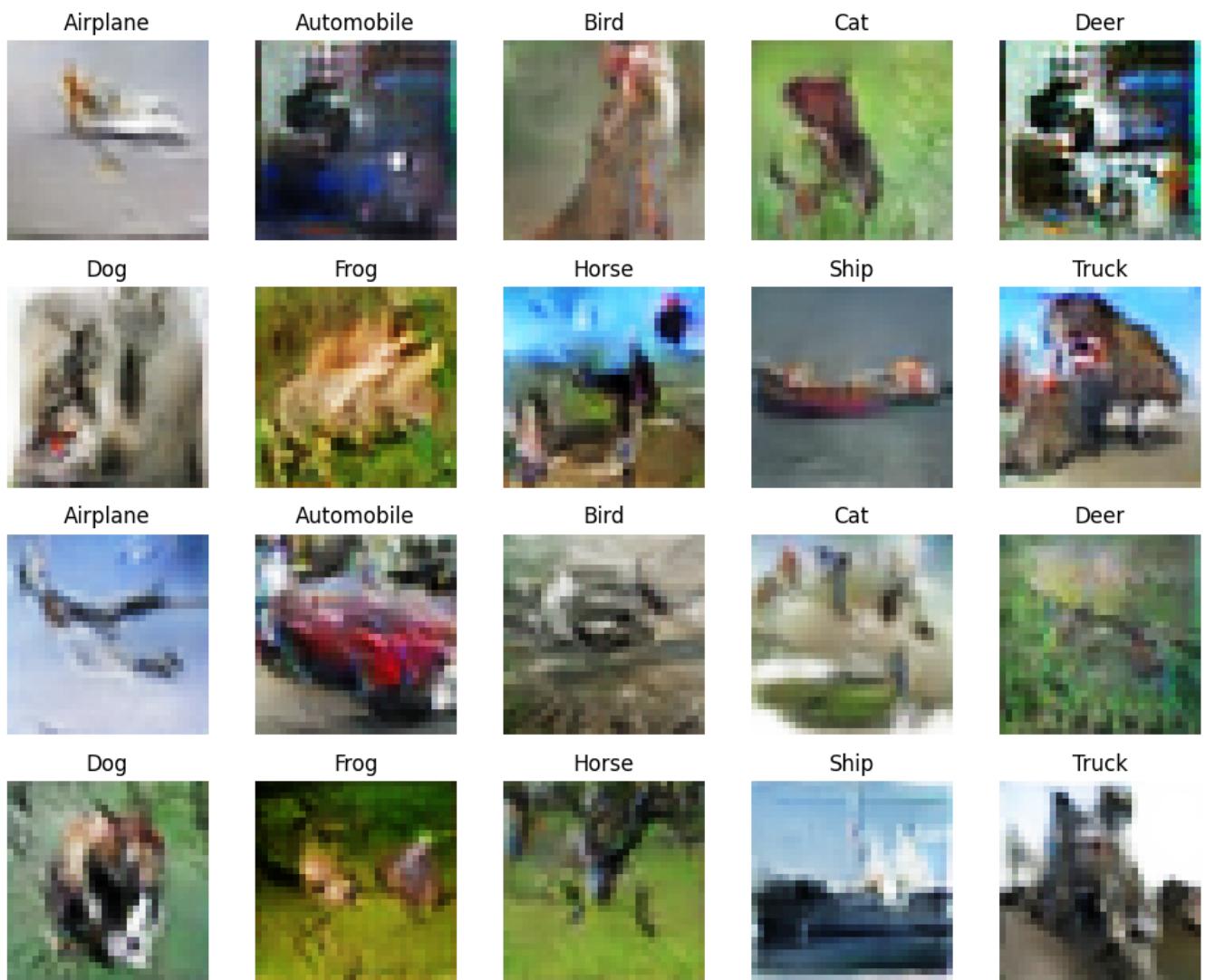
Epoch 84/200

782/782 [=====] - 56s 72ms/step - d_loss: 0.1138 - g_loss: 6.2236 - D(x|y): 0.5003 - D(G(z|y)): 0.0604 - KL Divergence: 4.5212

Epoch 85/200

782/782 [=====] - 56s 72ms/step - d_loss: 0.1212 - g_loss: 6.1302 - D(x|y): 0.4998 - D(G(z|y)): 0.0597 - KL Divergence: 4.5620

1/1 [=====] - 0s 37ms/step



Generator Checkpoint - New cGAN/generator-epoch-85.h5

Epoch 86/200

782/782 [=====] - 58s 74ms/step - d_loss: 0.1112 - g_loss: 6.2619 -
 $D(x|y): 0.5004$ - $D(G(z|y)): 0.0577$ - KL Divergence: 4.7000

Epoch 87/200

782/782 [=====] - 65s 83ms/step - d_loss: 0.1115 - g_loss: 6.3115 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0604$ - KL Divergence: 4.7278

Epoch 88/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.1073 - g_loss: 6.3657 -
 $D(x|y): 0.5003$ - $D(G(z|y)): 0.0575$ - KL Divergence: 4.6410

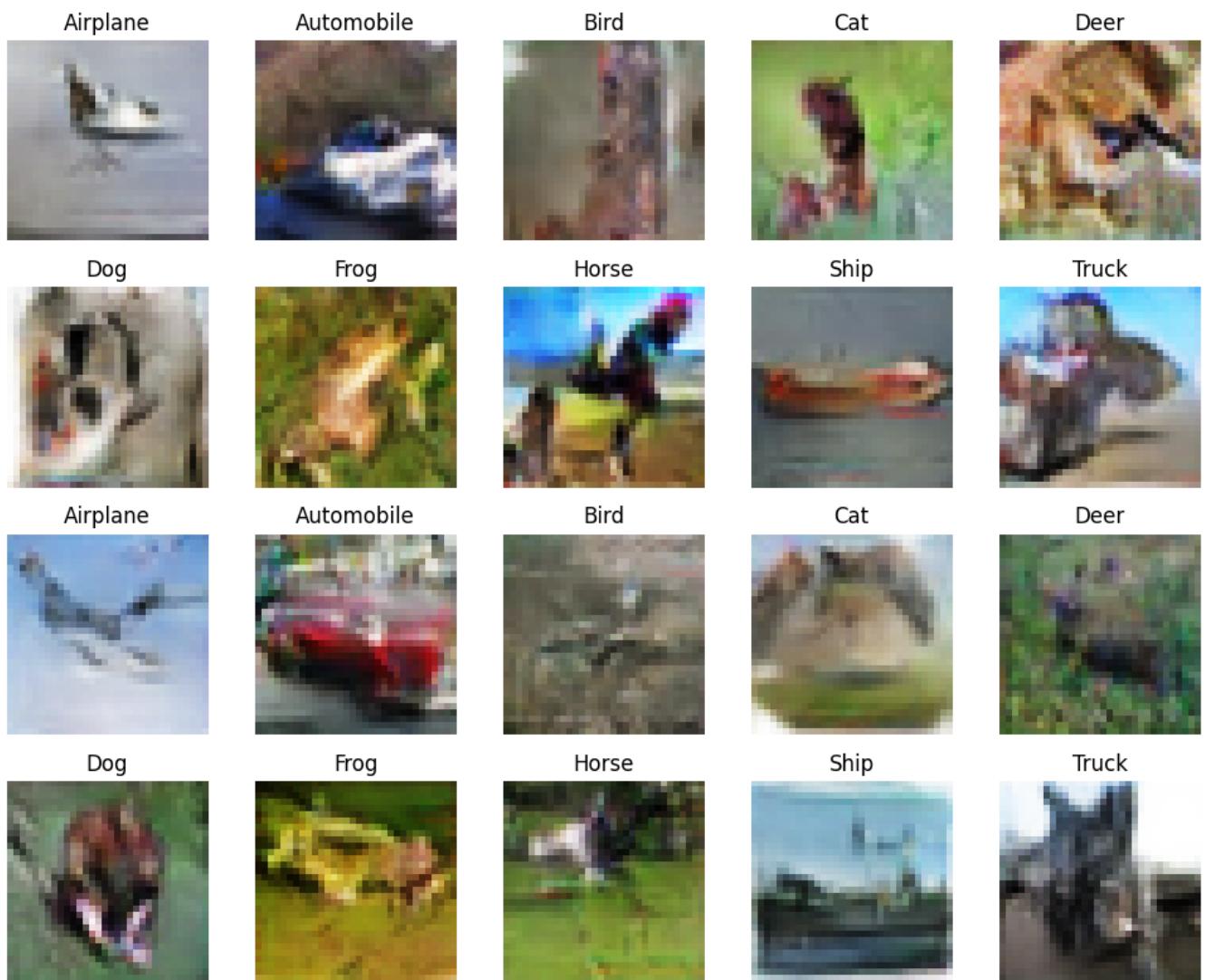
Epoch 89/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.1094 - g_loss: 6.3513 -
 $D(x|y): 0.5008$ - $D(G(z|y)): 0.0578$ - KL Divergence: 4.6047

Epoch 90/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.1095 - g_loss: 6.3270 -
 $D(x|y): 0.5002$ - $D(G(z|y)): 0.0589$ - KL Divergence: 4.5860

1/1 [=====] - 0s 51ms/step



Generator Checkpoint - New cGAN/generator-epoch-90.h5

Epoch 91/200

782/782 [=====] - 80s 102ms/step - d_loss: 0.1058 - g_loss: 6.4651 - D(x|y): 0.4999 - D(G(z|y)): 0.0575 - KL Divergence: 4.5938

Epoch 92/200

782/782 [=====] - 80s 102ms/step - d_loss: 0.1043 - g_loss: 6.4436 - D(x|y): 0.5003 - D(G(z|y)): 0.0581 - KL Divergence: 4.5124

Epoch 93/200

782/782 [=====] - 80s 102ms/step - d_loss: 0.1143 - g_loss: 6.3768 - D(x|y): 0.5003 - D(G(z|y)): 0.0608 - KL Divergence: 4.5867

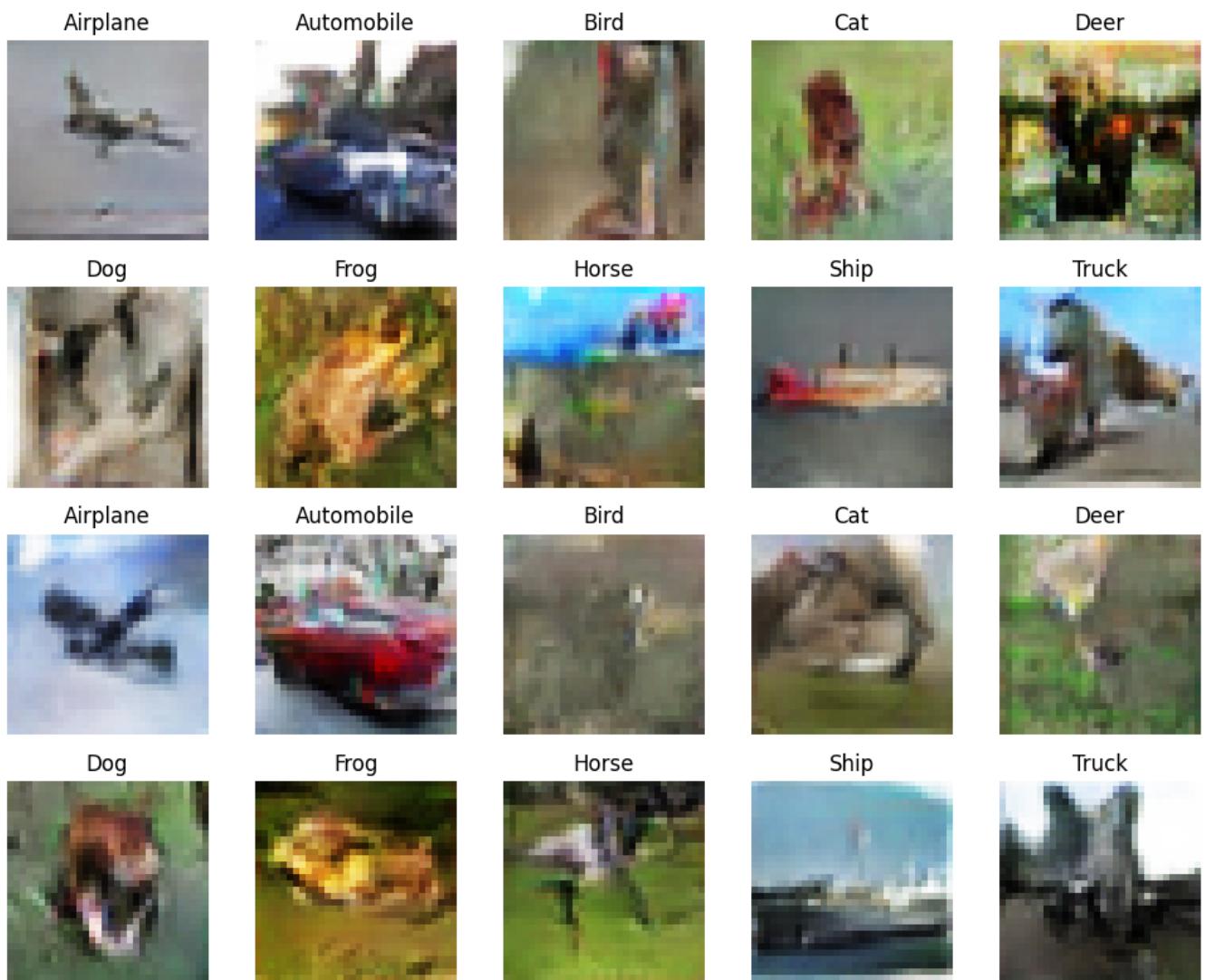
Epoch 94/200

782/782 [=====] - 80s 103ms/step - d_loss: 0.1055 - g_loss: 6.4705 - D(x|y): 0.5005 - D(G(z|y)): 0.0570 - KL Divergence: 4.6591

Epoch 95/200

782/782 [=====] - 83s 106ms/step - d_loss: 0.1087 - g_loss: 6.4679 - D(x|y): 0.5003 - D(G(z|y)): 0.0570 - KL Divergence: 4.5465

1/1 [=====] - 0s 82ms/step



Generator Checkpoint - New cGAN/generator-epoch-95.h5

Epoch 96/200

782/782 [=====] - 87s 111ms/step - d_loss: 0.1072 - g_loss: 6.5301 - D(x|y): 0.5007 - D(G(z|y)): 0.0565 - KL Divergence: 4.6018

Epoch 97/200

782/782 [=====] - 90s 116ms/step - d_loss: 0.1100 - g_loss: 6.6123 - D(x|y): 0.5001 - D(G(z|y)): 0.0557 - KL Divergence: 4.6057

Epoch 98/200

782/782 [=====] - 86s 109ms/step - d_loss: 0.1081 - g_loss: 6.5173 - D(x|y): 0.5004 - D(G(z|y)): 0.0556 - KL Divergence: 4.7520

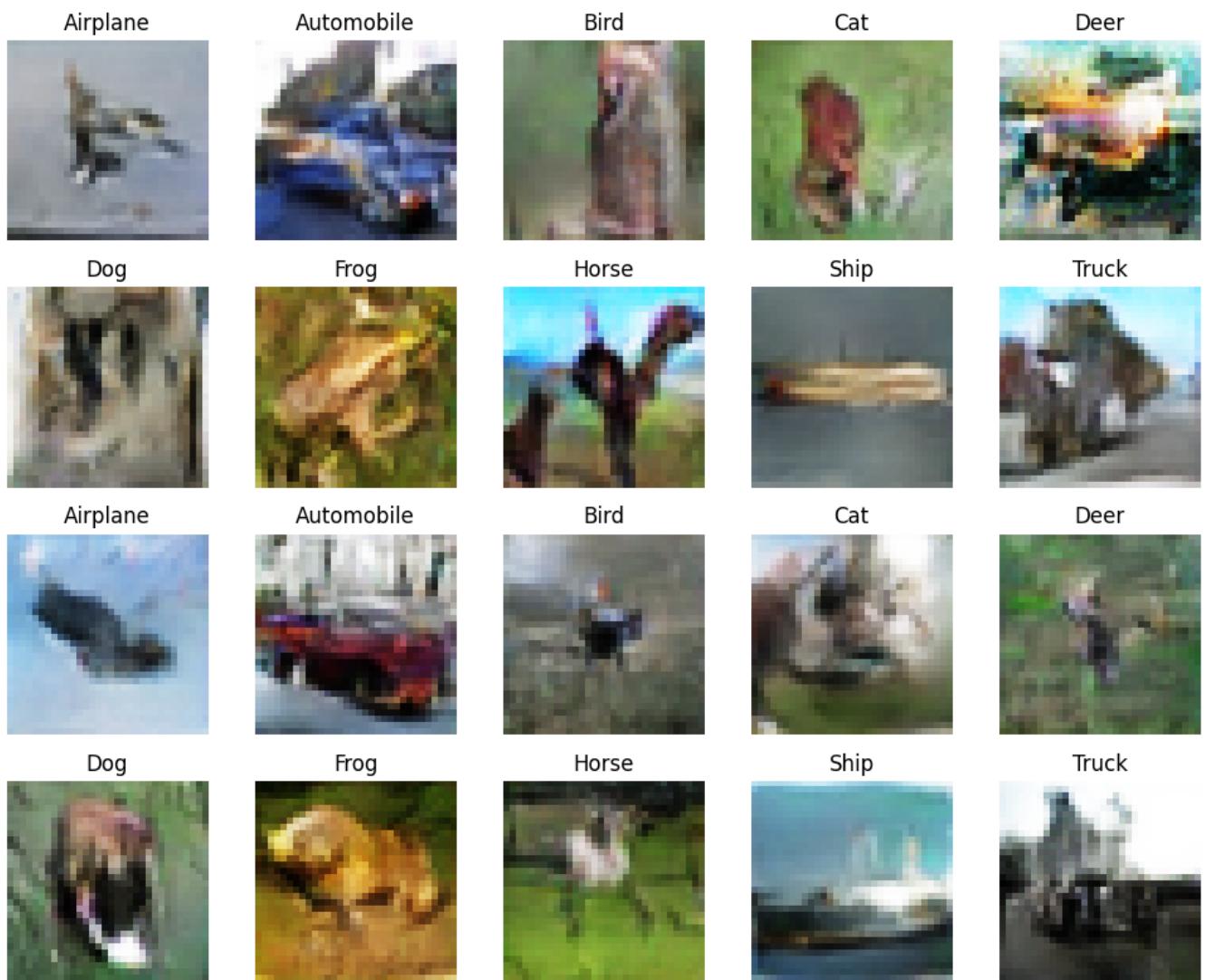
Epoch 99/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.1057 - g_loss: 6.5616 - D(x|y): 0.5002 - D(G(z|y)): 0.0549 - KL Divergence: 4.6622

Epoch 100/200

782/782 [=====] - 82s 105ms/step - d_loss: 0.1048 - g_loss: 6.6062 - D(x|y): 0.4998 - D(G(z|y)): 0.0541 - KL Divergence: 4.6291

1/1 [=====] - 0s 59ms/step



Generator Checkpoint - New cGAN/generator-epoch-100.h5

Epoch 101/200

782/782 [=====] - 82s 105ms/step - d_loss: 0.1006 - g_loss: 6.6132 -
 $D(x|y)$: 0.5010 - $D(G(z|y))$: 0.0528 - KL Divergence: 4.6096

Epoch 102/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.1058 - g_loss: 6.6935 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0540 - KL Divergence: 4.7265

Epoch 103/200

782/782 [=====] - 82s 105ms/step - d_loss: 0.0948 - g_loss: 6.7738 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0523 - KL Divergence: 4.7126

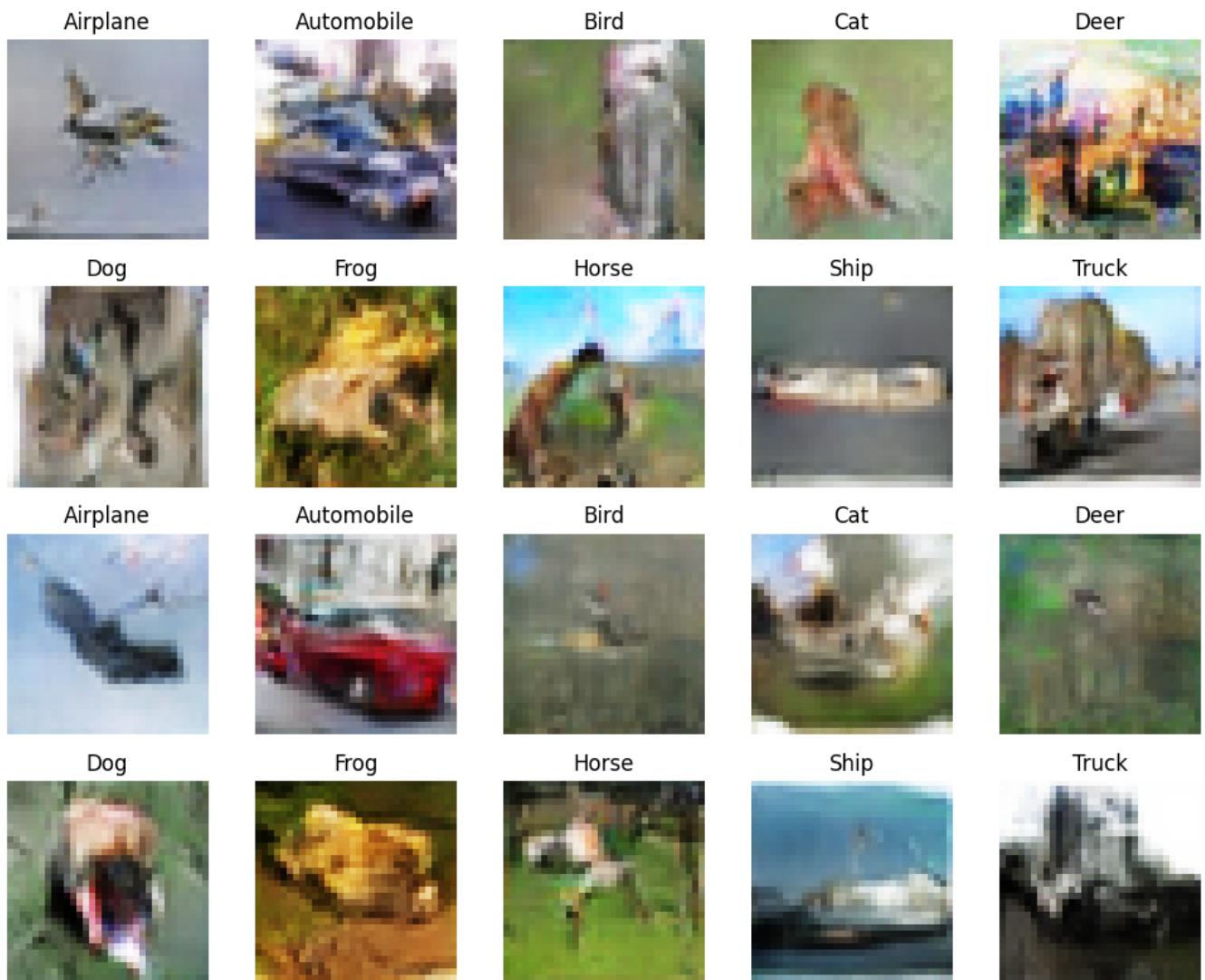
Epoch 104/200

782/782 [=====] - 82s 105ms/step - d_loss: 0.1006 - g_loss: 6.7151 -
 $D(x|y)$: 0.5009 - $D(G(z|y))$: 0.0521 - KL Divergence: 4.5973

Epoch 105/200

782/782 [=====] - 82s 104ms/step - d_loss: 0.1090 - g_loss: 6.7528 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0547 - KL Divergence: 4.6664

1/1 [=====] - 0s 59ms/step



Generator Checkpoint - New cGAN/generator-epoch-105.h5

Epoch 106/200

782/782 [=====] - 81s 104ms/step - d_loss: 0.1080 - g_loss: 6.6742 - D(x|y): 0.5003 - D(G(z|y)): 0.0553 - KL Divergence: 4.5811

Epoch 107/200

782/782 [=====] - 82s 105ms/step - d_loss: 0.1012 - g_loss: 6.8466 - D(x|y): 0.5003 - D(G(z|y)): 0.0510 - KL Divergence: 4.7239

Epoch 108/200

782/782 [=====] - 82s 104ms/step - d_loss: 0.0990 - g_loss: 6.7692 - D(x|y): 0.5000 - D(G(z|y)): 0.0499 - KL Divergence: 4.5766

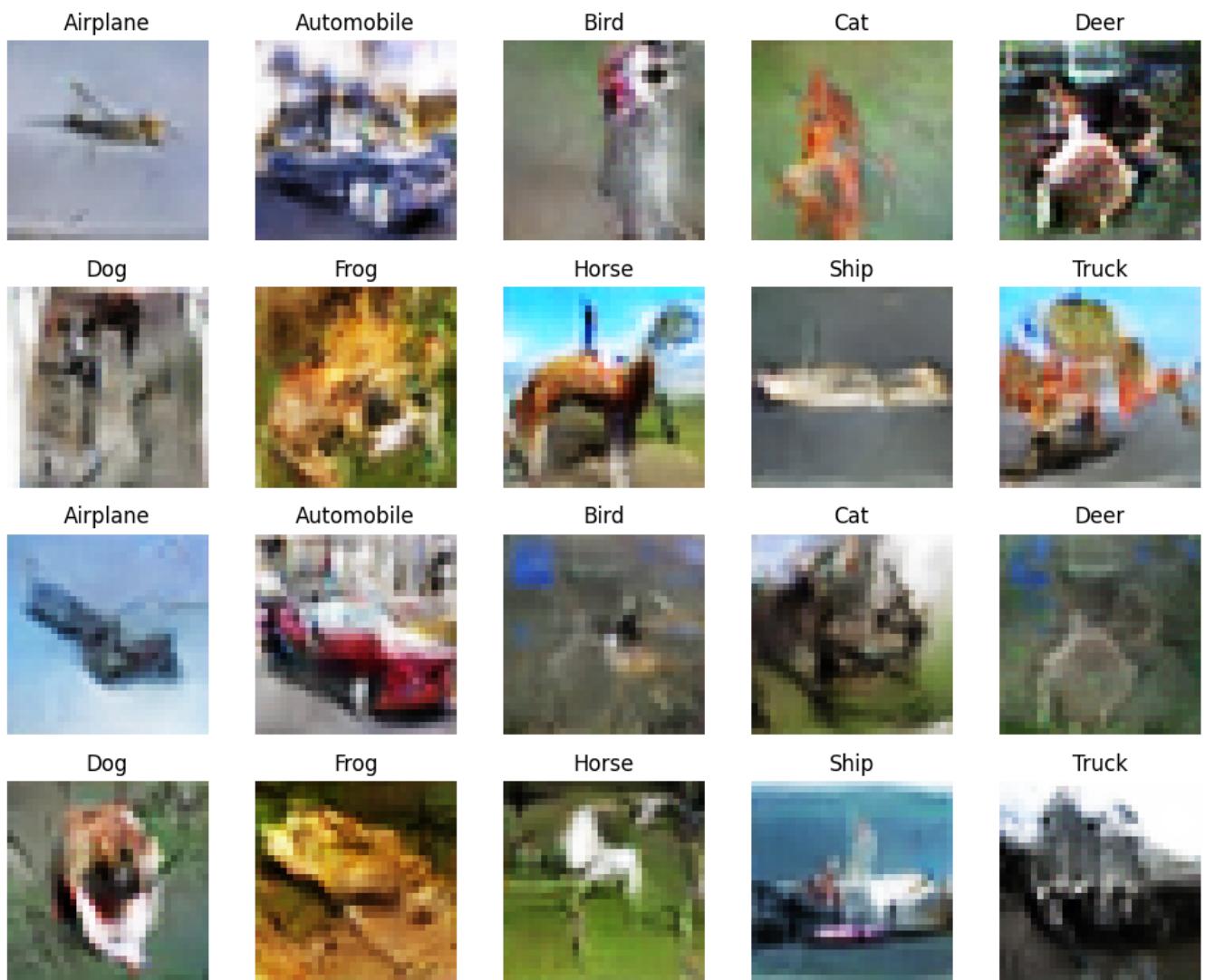
Epoch 109/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.1039 - g_loss: 6.8116 - D(x|y): 0.5000 - D(G(z|y)): 0.0506 - KL Divergence: 4.5668

Epoch 110/200

782/782 [=====] - 81s 103ms/step - d_loss: 0.0994 - g_loss: 6.8772 - D(x|y): 0.5003 - D(G(z|y)): 0.0514 - KL Divergence: 4.6859

1/1 [=====] - 0s 49ms/step



Generator Checkpoint - New cGAN/generator-epoch-110.h5

Epoch 111/200

782/782 [=====] - 82s 105ms/step - d_loss: 0.0944 - g_loss: 6.9157 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0489 - KL Divergence: 4.6663

Epoch 112/200

782/782 [=====] - 66s 84ms/step - d_loss: 0.0959 - g_loss: 6.9419 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0495 - KL Divergence: 4.6097

Epoch 113/200

782/782 [=====] - 45s 58ms/step - d_loss: 0.0985 - g_loss: 7.0470 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0492 - KL Divergence: 4.4436

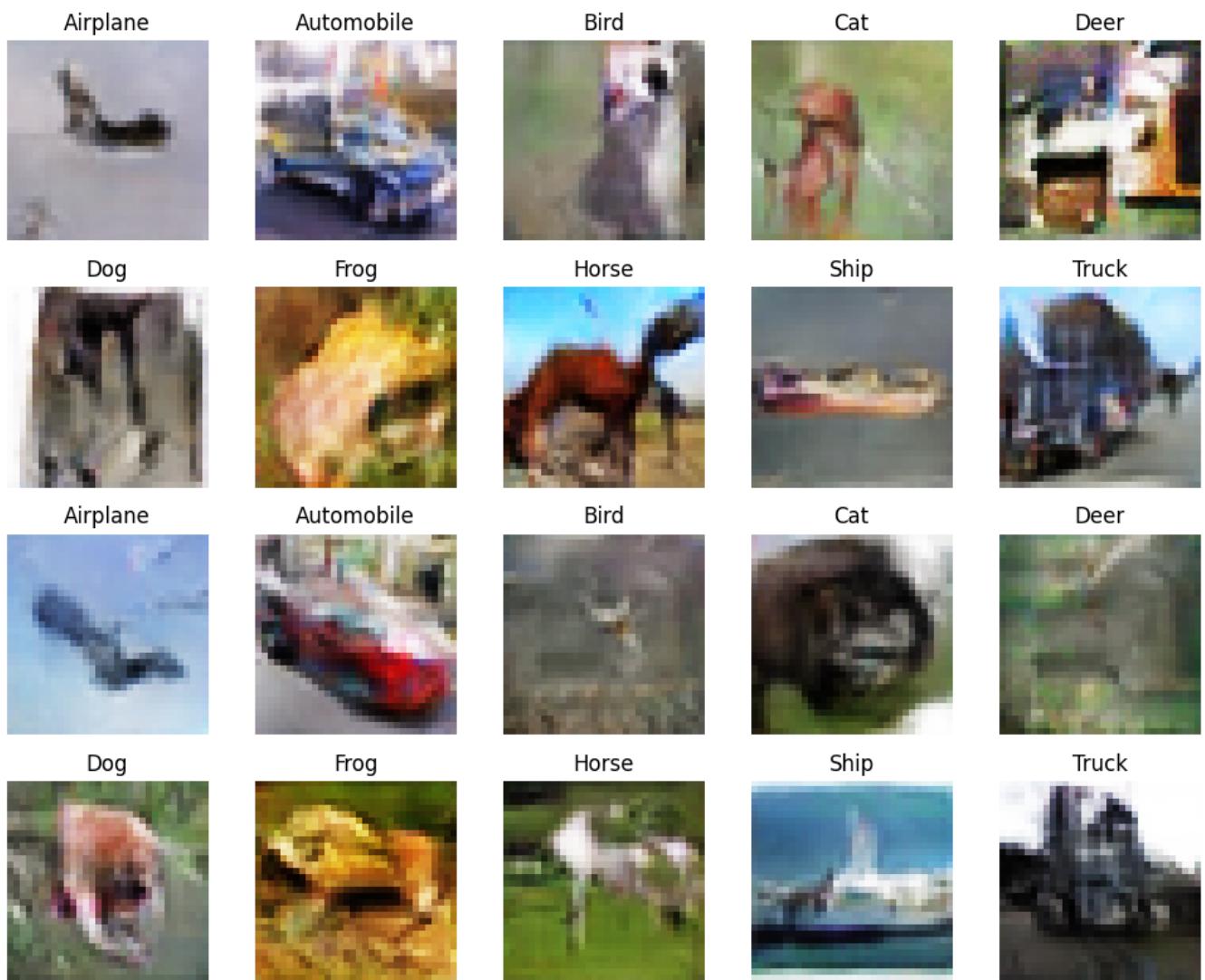
Epoch 114/200

782/782 [=====] - 45s 58ms/step - d_loss: 0.0974 - g_loss: 7.0387 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0494 - KL Divergence: 4.5173

Epoch 115/200

782/782 [=====] - 46s 58ms/step - d_loss: 0.0990 - g_loss: 6.9390 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0510 - KL Divergence: 4.5969

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New cGAN/generator-epoch-115.h5

Epoch 116/200

782/782 [=====] - 45s 57ms/step - d_loss: 0.0993 - g_loss: 7.0117 -
 $D(x|y)$: 0.5012 - $D(G(z|y))$: 0.0513 - KL Divergence: 4.6055

Epoch 117/200

782/782 [=====] - 45s 57ms/step - d_loss: 0.0929 - g_loss: 7.1218 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0481 - KL Divergence: 4.6520

Epoch 118/200

782/782 [=====] - 45s 58ms/step - d_loss: 0.0956 - g_loss: 7.0729 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0496 - KL Divergence: 4.6151

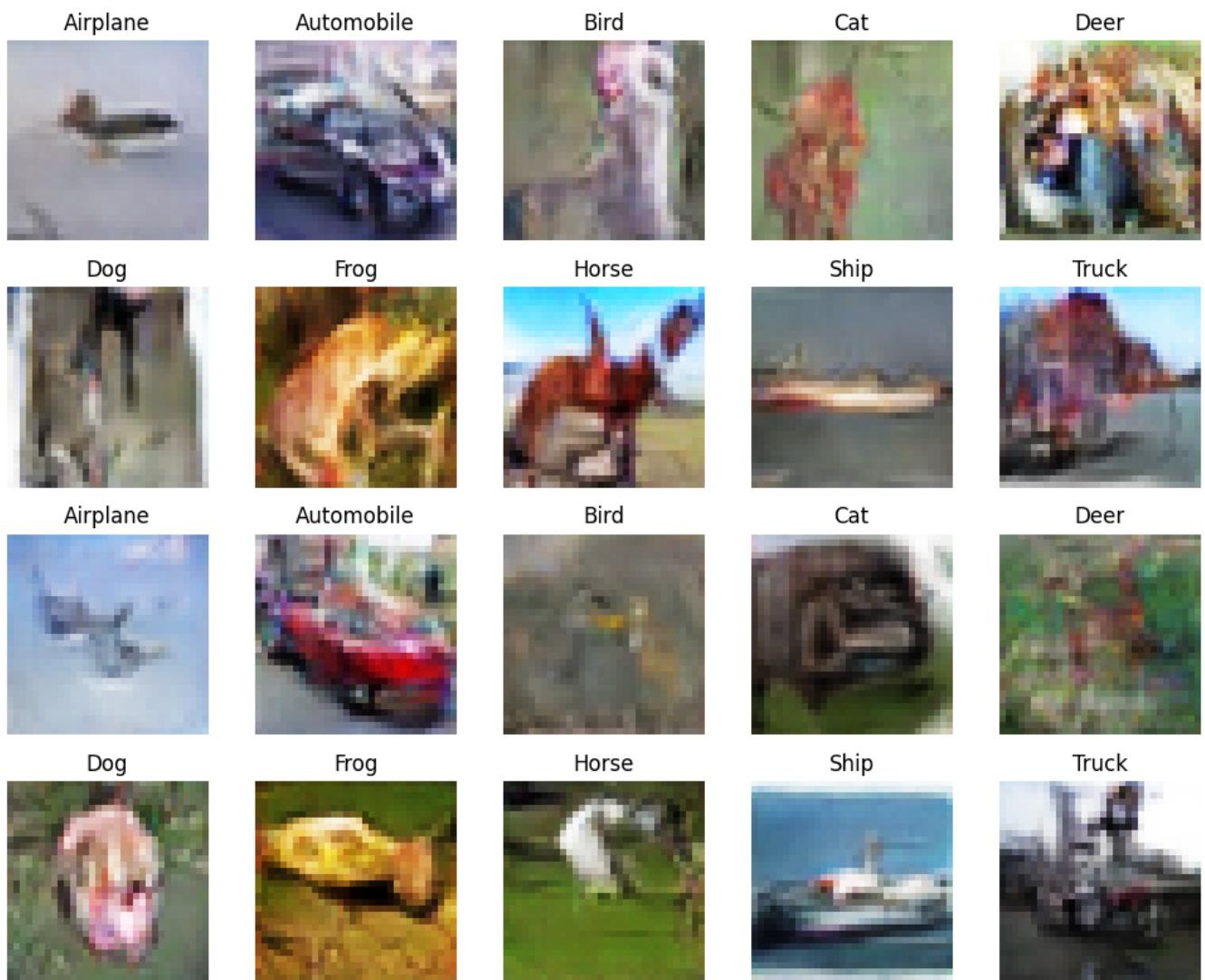
Epoch 119/200

782/782 [=====] - 51s 65ms/step - d_loss: 0.0941 - g_loss: 7.0706 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0495 - KL Divergence: 4.5493

Epoch 120/200

782/782 [=====] - 51s 66ms/step - d_loss: 0.0879 - g_loss: 7.2113 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0457 - KL Divergence: 4.5559

1/1 [=====] - 0s 34ms/step



Generator Checkpoint - New cGAN/generator-epoch-120.h5

Epoch 121/200

782/782 [=====] - 52s 67ms/step - d_loss: 0.0961 - g_loss: 7.1849 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0485 - KL Divergence: 4.6121

Epoch 122/200

782/782 [=====] - 49s 63ms/step - d_loss: 0.0906 - g_loss: 7.2247 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0481 - KL Divergence: 4.5534

Epoch 123/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0936 - g_loss: 7.2385 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0454 - KL Divergence: 4.6024

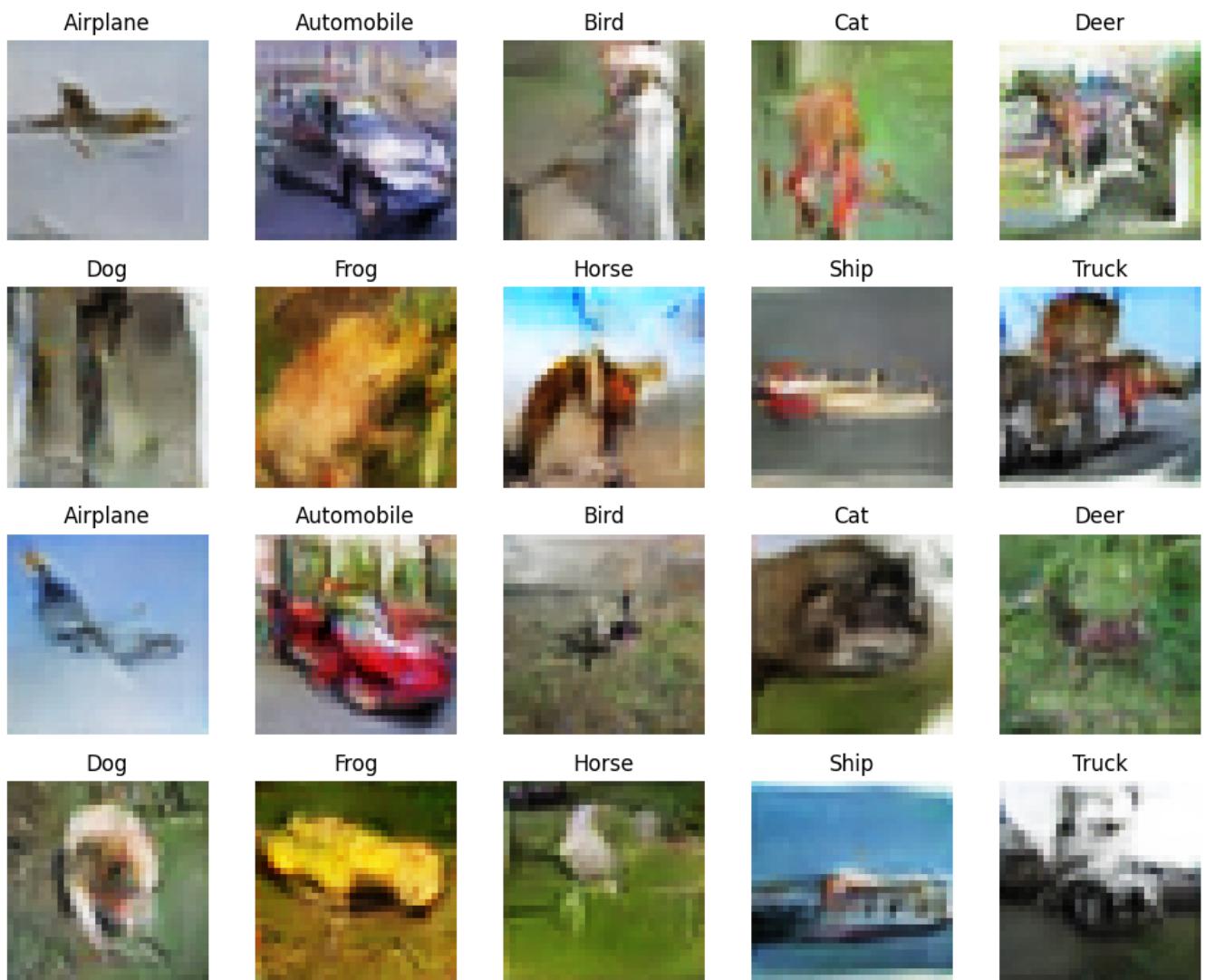
Epoch 124/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0815 - g_loss: 7.3134 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0442 - KL Divergence: 4.6128

Epoch 125/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0886 - g_loss: 7.3857 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0458 - KL Divergence: 4.7411

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - New cGAN/generator-epoch-125.h5

Epoch 126/200

782/782 [=====] - 49s 62ms/step - d_loss: 0.0898 - g_loss: 7.2204 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0466 - KL Divergence: 4.3763

Epoch 127/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0904 - g_loss: 7.2214 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0476 - KL Divergence: 4.5590

Epoch 128/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0883 - g_loss: 7.3766 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.0451 - KL Divergence: 4.5205

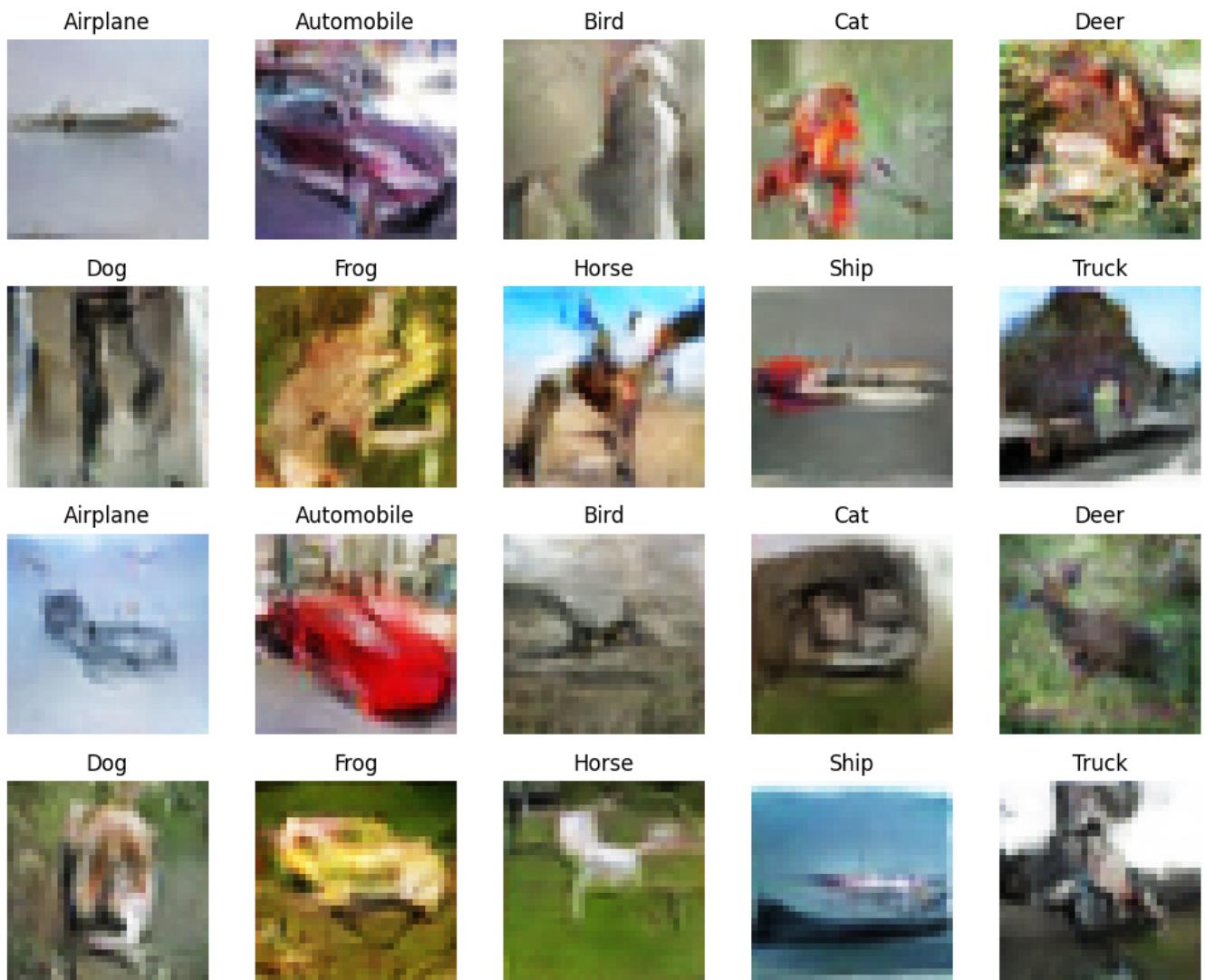
Epoch 129/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0865 - g_loss: 7.4122 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0446 - KL Divergence: 4.4949

Epoch 130/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0905 - g_loss: 7.3925 -
 $D(x|y)$: 0.5010 - $D(G(z|y))$: 0.0476 - KL Divergence: 4.5745

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - New cGAN/generator-epoch-130.h5

Epoch 131/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0924 - g_loss: 7.3476 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0461 - KL Divergence: 4.5651

Epoch 132/200

782/782 [=====] - 48s 61ms/step - d_loss: 0.0846 - g_loss: 7.5804 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0416 - KL Divergence: 4.6822

Epoch 133/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0896 - g_loss: 7.5211 -
 $D(x|y)$: 0.4998 - $D(G(z|y))$: 0.0449 - KL Divergence: 4.6315

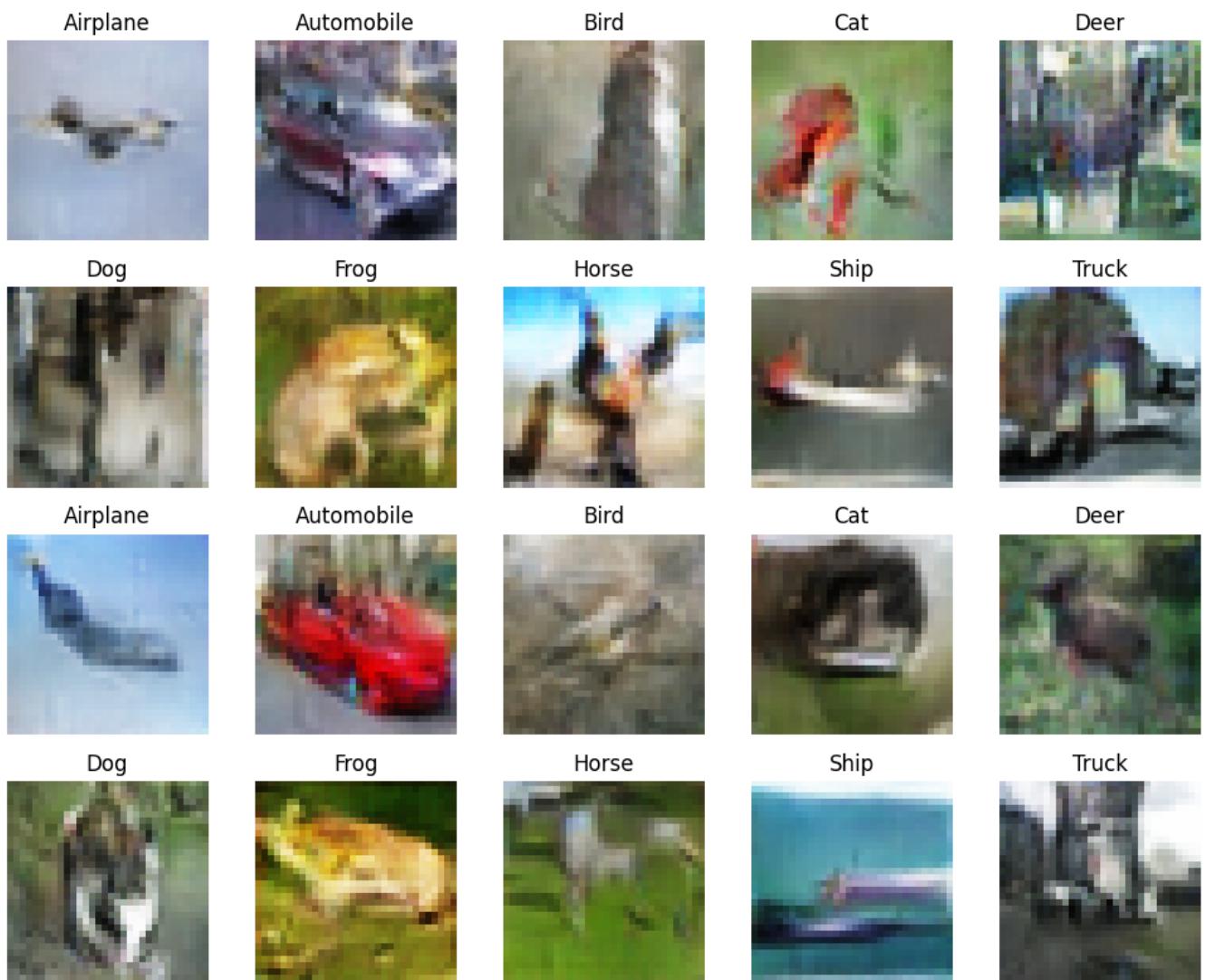
Epoch 134/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0882 - g_loss: 7.4527 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0465 - KL Divergence: 4.5537

Epoch 135/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0880 - g_loss: 7.5451 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0441 - KL Divergence: 4.6983

1/1 [=====] - 0s 29ms/step



Generator Checkpoint - New cGAN/generator-epoch-135.h5

Epoch 136/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0824 - g_loss: 7.6167 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0428 - KL Divergence: 4.6904

Epoch 137/200

782/782 [=====] - 49s 62ms/step - d_loss: 0.0813 - g_loss: 7.6924 -
 $D(x|y)$: 0.5009 - $D(G(z|y))$: 0.0428 - KL Divergence: 4.7189

Epoch 138/200

782/782 [=====] - 49s 62ms/step - d_loss: 0.0868 - g_loss: 7.6051 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0422 - KL Divergence: 4.6432

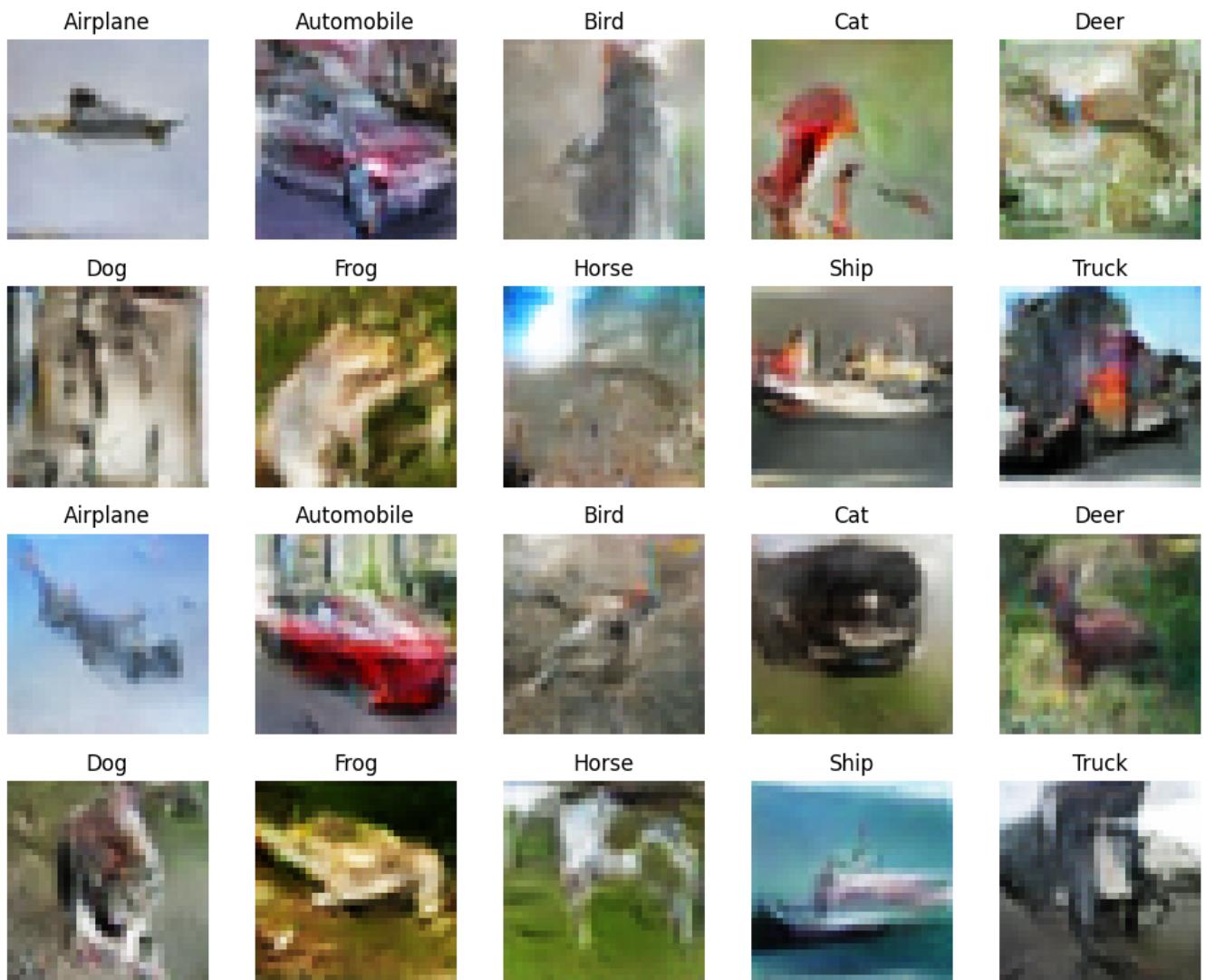
Epoch 139/200

782/782 [=====] - 49s 62ms/step - d_loss: 0.0889 - g_loss: 7.5492 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0466 - KL Divergence: 4.4963

Epoch 140/200

782/782 [=====] - 49s 62ms/step - d_loss: 0.0835 - g_loss: 7.5683 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0416 - KL Divergence: 4.5735

1/1 [=====] - 0s 31ms/step



Generator Checkpoint - New cGAN/generator-epoch-140.h5

Epoch 141/200

782/782 [=====] - 49s 63ms/step - d_loss: 0.0883 - g_loss: 7.5399 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0443 - KL Divergence: 4.4941

Epoch 142/200

782/782 [=====] - 48s 62ms/step - d_loss: 0.0847 - g_loss: 7.7248 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0422 - KL Divergence: 4.5994

Epoch 143/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0837 - g_loss: 7.6992 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0402 - KL Divergence: 4.6784

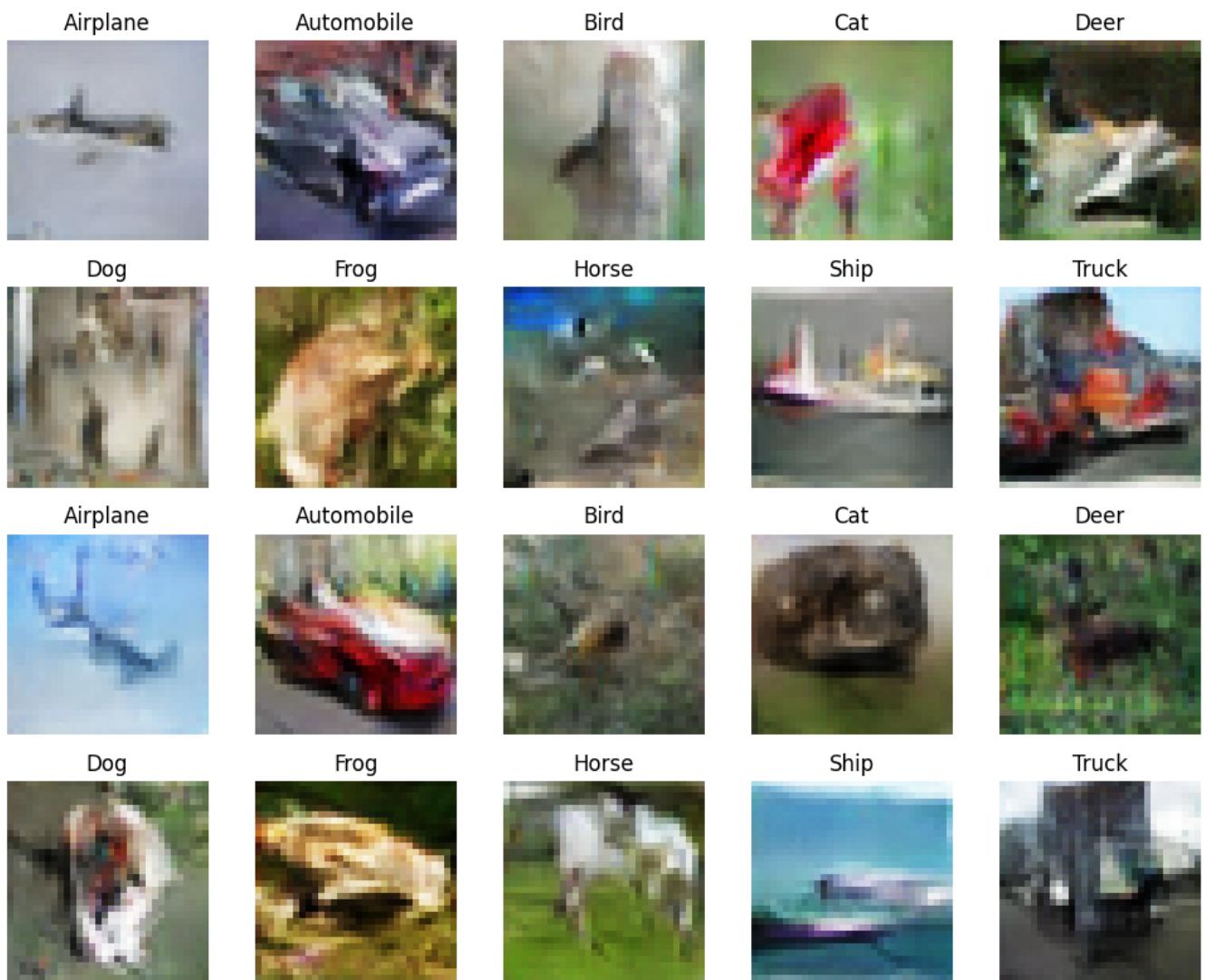
Epoch 144/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0853 - g_loss: 7.6384 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0440 - KL Divergence: 4.6025

Epoch 145/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0865 - g_loss: 7.7462 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0417 - KL Divergence: 4.6686

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New cGAN/generator-epoch-145.h5

Epoch 146/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0772 - g_loss: 7.9372 - D(x|y): 0.4997 - D(G(z|y)): 0.0376 - KL Divergence: 4.7923

Epoch 147/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0808 - g_loss: 7.8268 - D(x|y): 0.5000 - D(G(z|y)): 0.0412 - KL Divergence: 4.6729

Epoch 148/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0890 - g_loss: 7.6309 - D(x|y): 0.5005 - D(G(z|y)): 0.0432 - KL Divergence: 4.5258

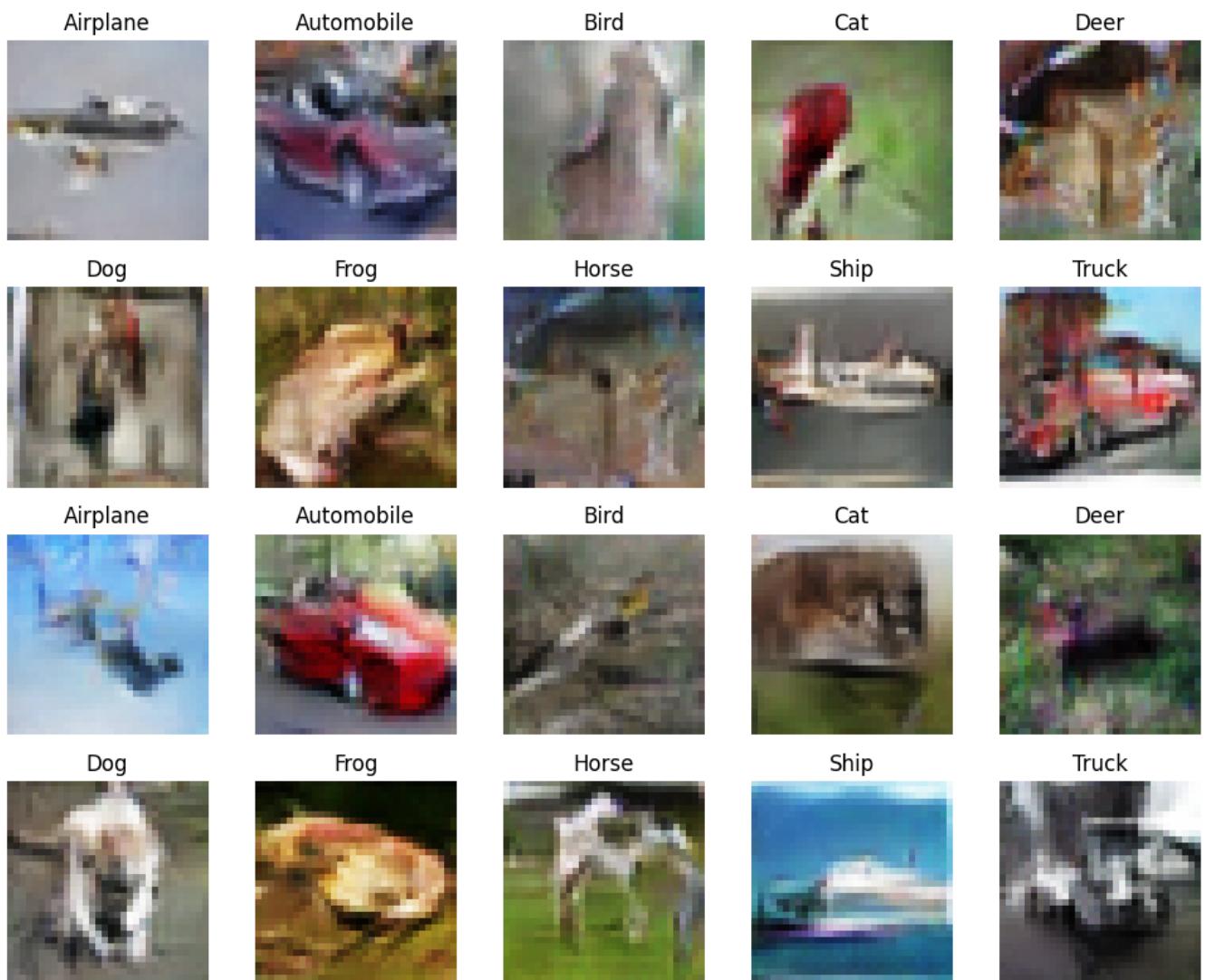
Epoch 149/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0834 - g_loss: 7.6751 - D(x|y): 0.5005 - D(G(z|y)): 0.0414 - KL Divergence: 4.6080

Epoch 150/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0815 - g_loss: 7.8251 - D(x|y): 0.5002 - D(G(z|y)): 0.0390 - KL Divergence: 4.8169

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New cGAN/generator-epoch-150.h5

Epoch 151/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0843 - g_loss: 7.8169 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0416 - KL Divergence: 4.7616

Epoch 152/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0826 - g_loss: 7.7697 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0405 - KL Divergence: 4.5633

Epoch 153/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0780 - g_loss: 7.8280 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0376 - KL Divergence: 4.5700

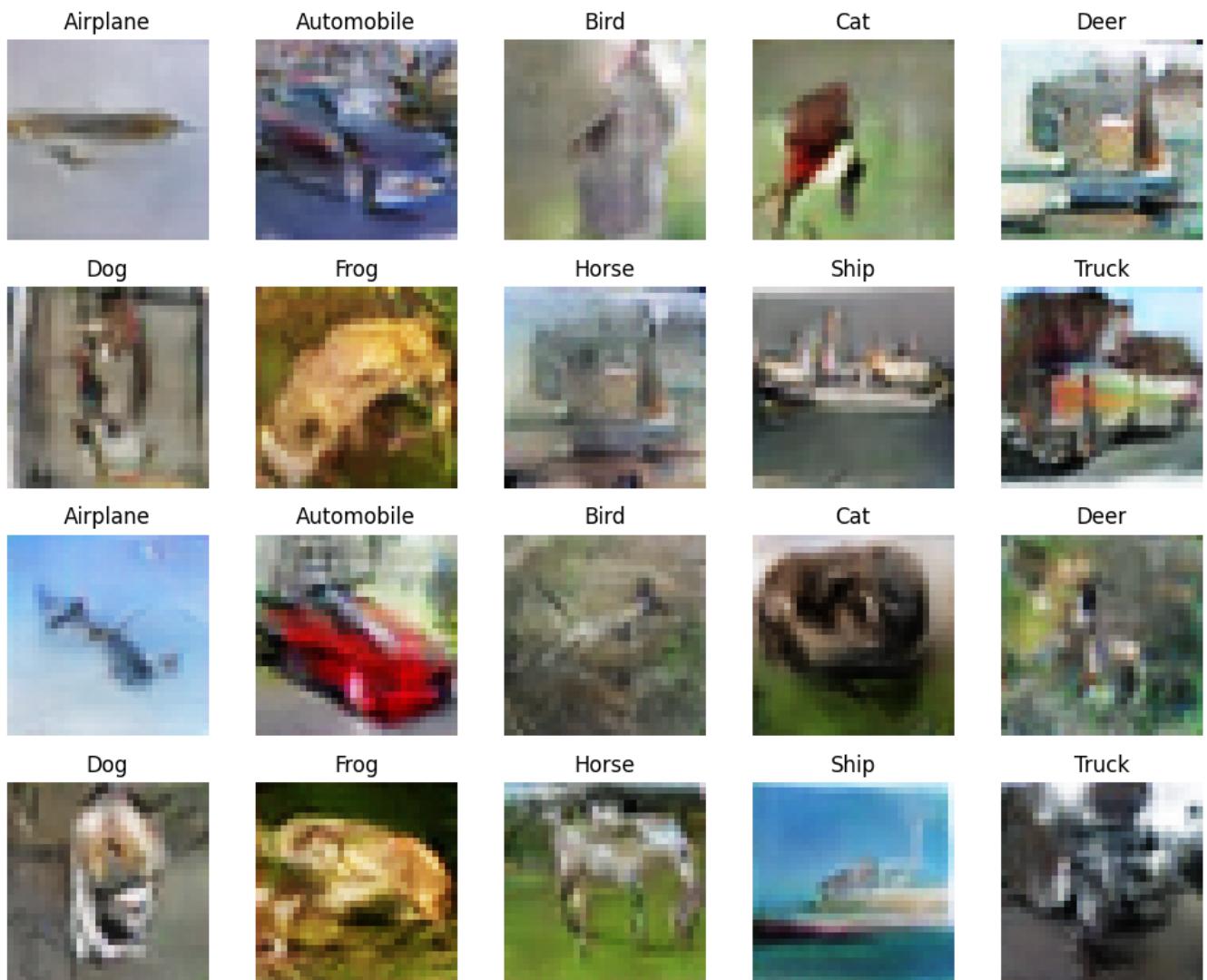
Epoch 154/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0858 - g_loss: 7.8240 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0410 - KL Divergence: 4.5518

Epoch 155/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0767 - g_loss: 7.8748 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0375 - KL Divergence: 4.5938

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New cGAN/generator-epoch-155.h5

Epoch 156/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0816 - g_loss: 7.9283 - D(x|y): 0.5005 - D(G(z|y)): 0.0385 - KL Divergence: 4.5177

Epoch 157/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0793 - g_loss: 8.0286 - D(x|y): 0.5005 - D(G(z|y)): 0.0378 - KL Divergence: 4.5369

Epoch 158/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0815 - g_loss: 7.8625 - D(x|y): 0.5001 - D(G(z|y)): 0.0405 - KL Divergence: 4.7889

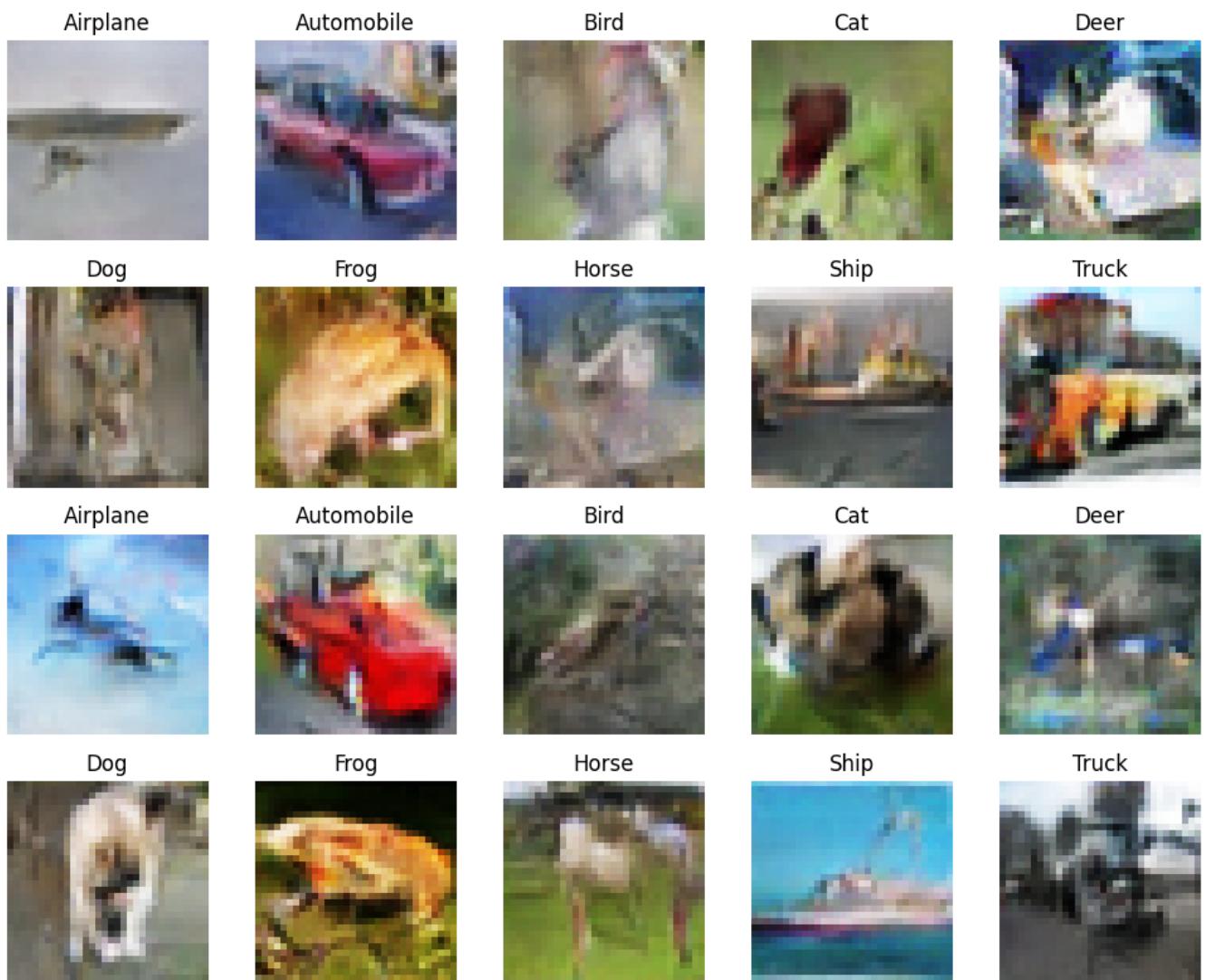
Epoch 159/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0794 - g_loss: 7.8124 - D(x|y): 0.4999 - D(G(z|y)): 0.0415 - KL Divergence: 4.5146

Epoch 160/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0757 - g_loss: 7.8585 - D(x|y): 0.4996 - D(G(z|y)): 0.0385 - KL Divergence: 4.2978

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New cGAN/generator-epoch-160.h5

Epoch 161/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0837 - g_loss: 7.9510 - D(x|y): 0.5000 - D(G(z|y)): 0.0387 - KL Divergence: 4.3841

Epoch 162/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0821 - g_loss: 8.1019 - D(x|y): 0.5002 - D(G(z|y)): 0.0391 - KL Divergence: 4.3928

Epoch 163/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0754 - g_loss: 8.1943 - D(x|y): 0.5000 - D(G(z|y)): 0.0361 - KL Divergence: 4.6872

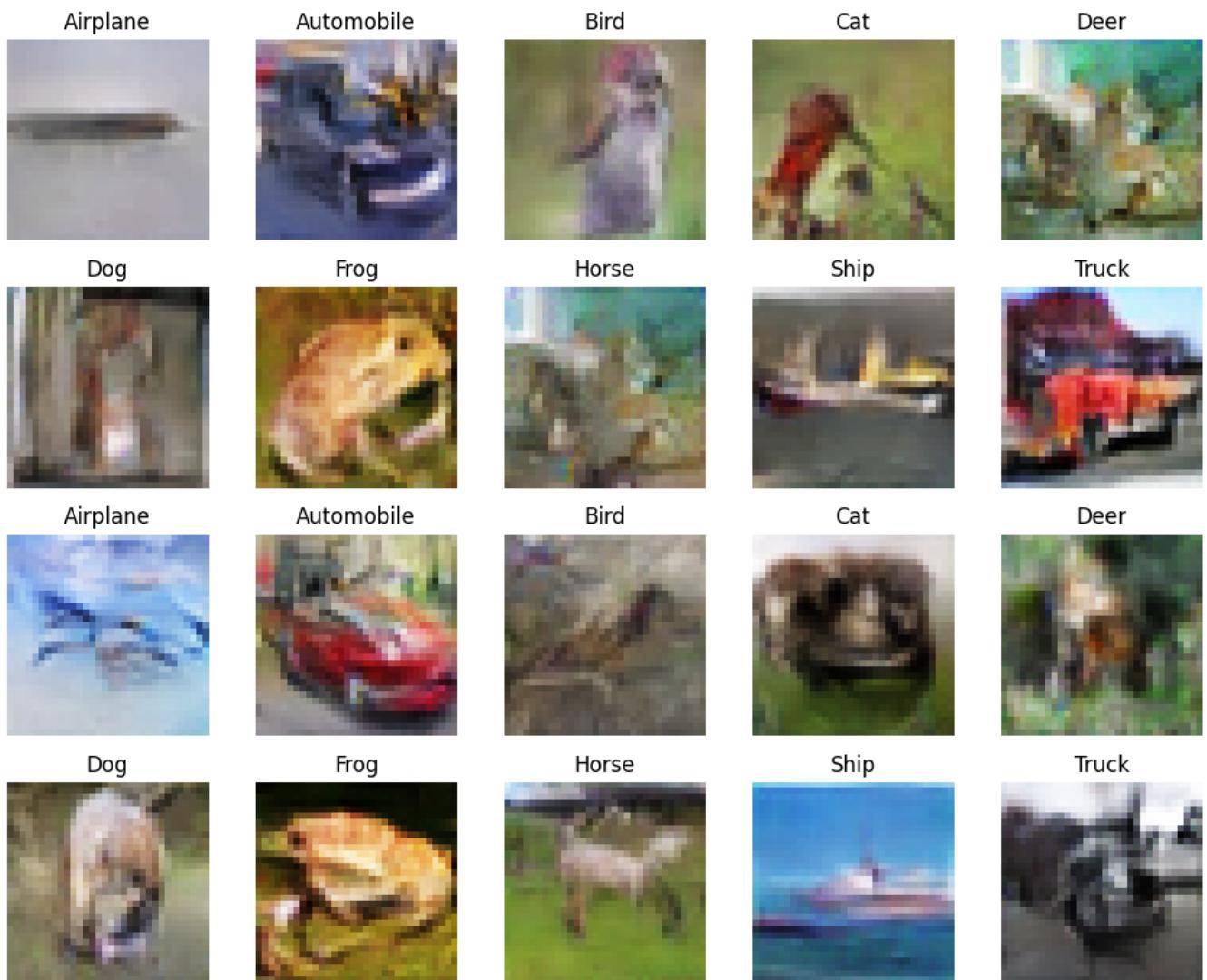
Epoch 164/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0689 - g_loss: 8.2363 - D(x|y): 0.5003 - D(G(z|y)): 0.0346 - KL Divergence: 4.7921

Epoch 165/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0709 - g_loss: 8.1513 - D(x|y): 0.5005 - D(G(z|y)): 0.0362 - KL Divergence: 4.6852

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New cGAN/generator-epoch-165.h5

Epoch 166/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0737 - g_loss: 8.0642 - D(x|y): 0.5005 - D(G(z|y)): 0.0379 - KL Divergence: 4.4787

Epoch 167/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0745 - g_loss: 8.0499 - D(x|y): 0.5003 - D(G(z|y)): 0.0389 - KL Divergence: 4.7231

Epoch 168/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0765 - g_loss: 7.9880 - D(x|y): 0.5003 - D(G(z|y)): 0.0364 - KL Divergence: 4.3611

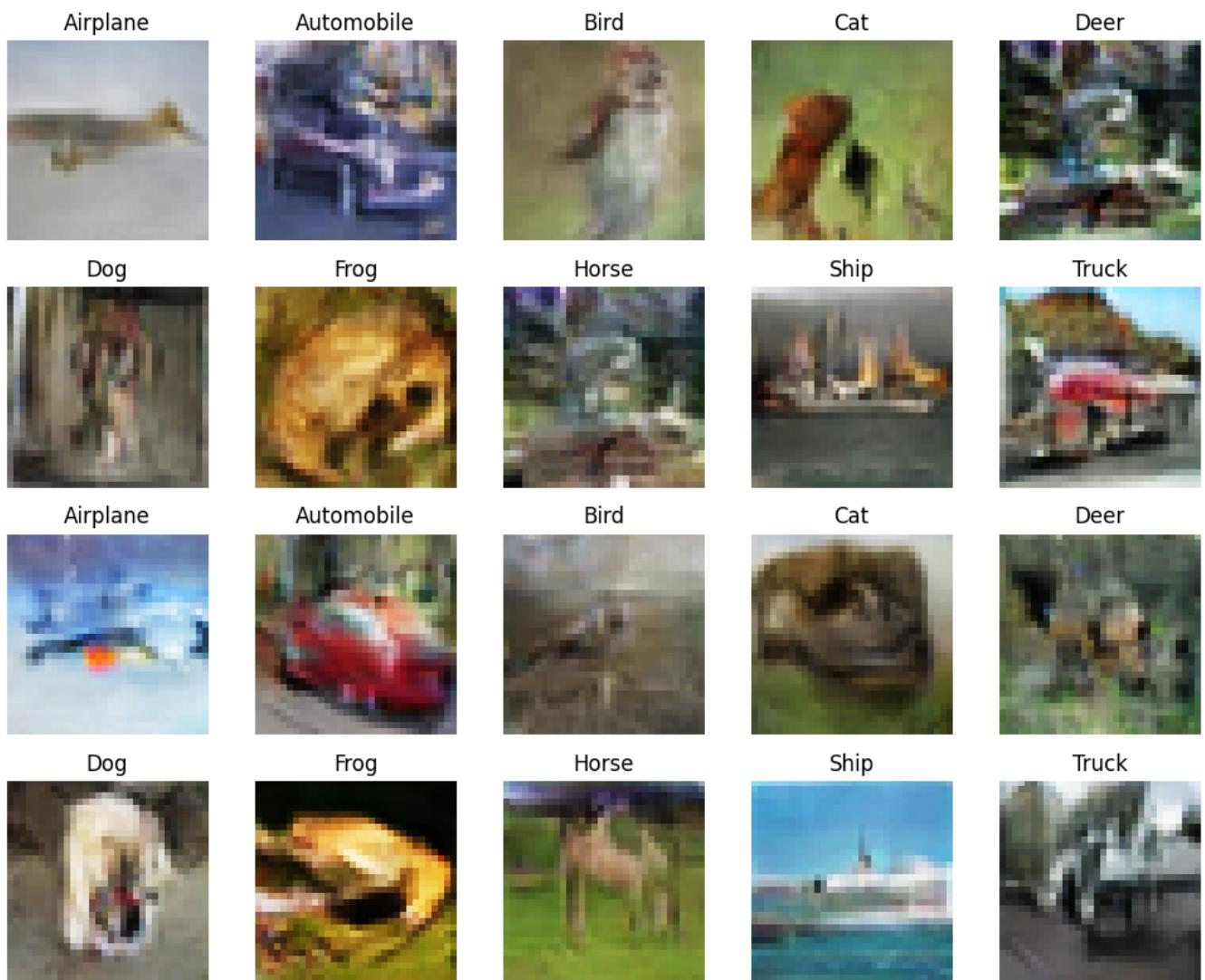
Epoch 169/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0775 - g_loss: 8.1135 - D(x|y): 0.5001 - D(G(z|y)): 0.0345 - KL Divergence: 4.2059

Epoch 170/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0764 - g_loss: 8.1005 - D(x|y): 0.4999 - D(G(z|y)): 0.0377 - KL Divergence: 4.4249

1/1 [=====] - 0s 27ms/step



Generator Checkpoint - New cGAN/generator-epoch-170.h5

Epoch 171/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0806 - g_loss: 8.3493 - D(x|y): 0.4998 - D(G(z|y)): 0.0366 - KL Divergence: 4.7445

Epoch 172/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0732 - g_loss: 8.1573 - D(x|y): 0.5001 - D(G(z|y)): 0.0365 - KL Divergence: 4.5028

Epoch 173/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0692 - g_loss: 8.3394 - D(x|y): 0.5004 - D(G(z|y)): 0.0344 - KL Divergence: 4.4728

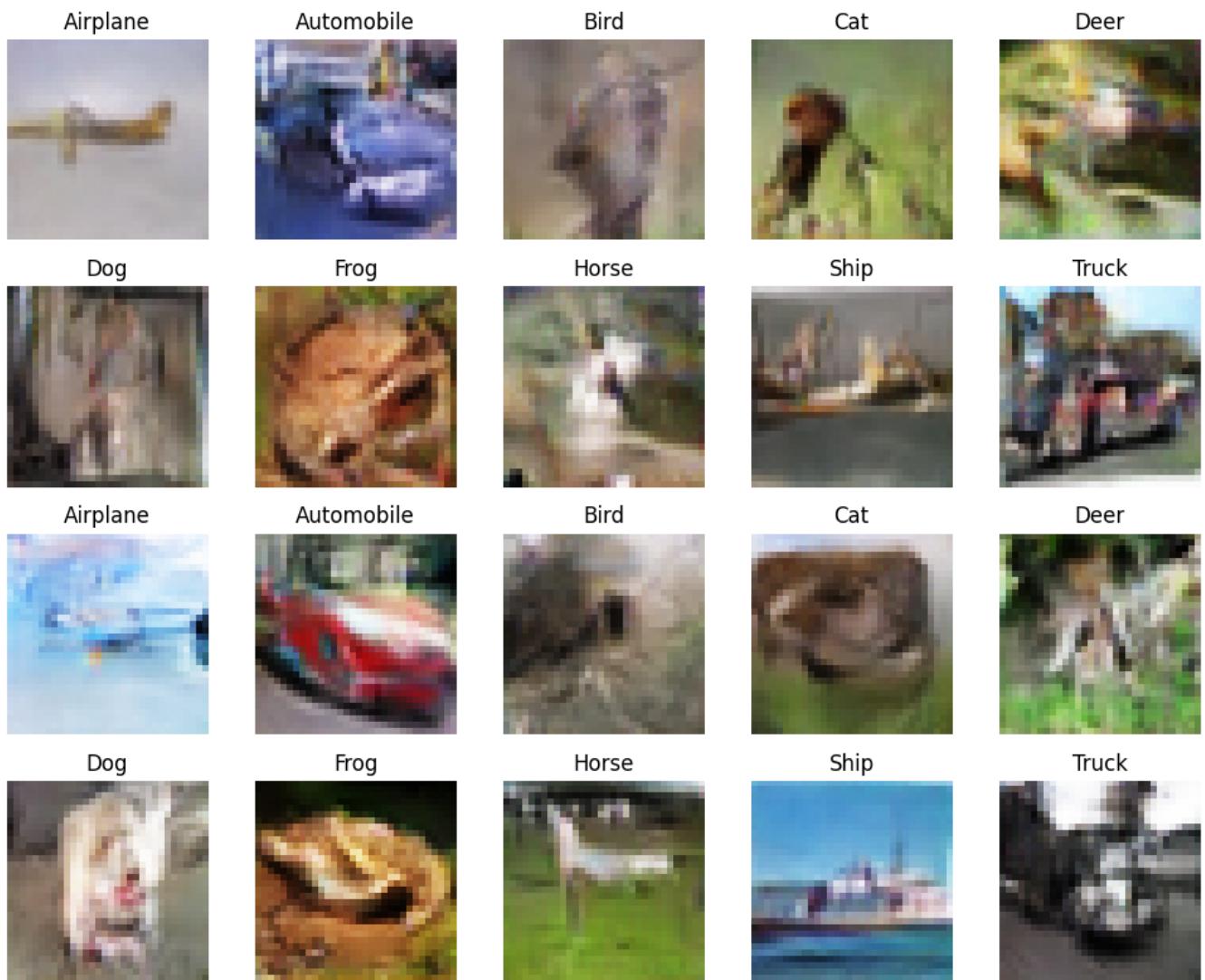
Epoch 174/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0700 - g_loss: 8.3384 - D(x|y): 0.5001 - D(G(z|y)): 0.0348 - KL Divergence: 4.3912

Epoch 175/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0719 - g_loss: 8.2288 - D(x|y): 0.4997 - D(G(z|y)): 0.0330 - KL Divergence: 4.3775

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New cGAN/generator-epoch-175.h5

Epoch 176/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0674 - g_loss: 8.3023 - D(x|y): 0.4999 - D(G(z|y)): 0.0342 - KL Divergence: 4.5305

Epoch 177/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0769 - g_loss: 8.0972 - D(x|y): 0.4992 - D(G(z|y)): 0.0367 - KL Divergence: 4.5238

Epoch 178/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0632 - g_loss: 8.4507 - D(x|y): 0.5000 - D(G(z|y)): 0.0325 - KL Divergence: 4.6332

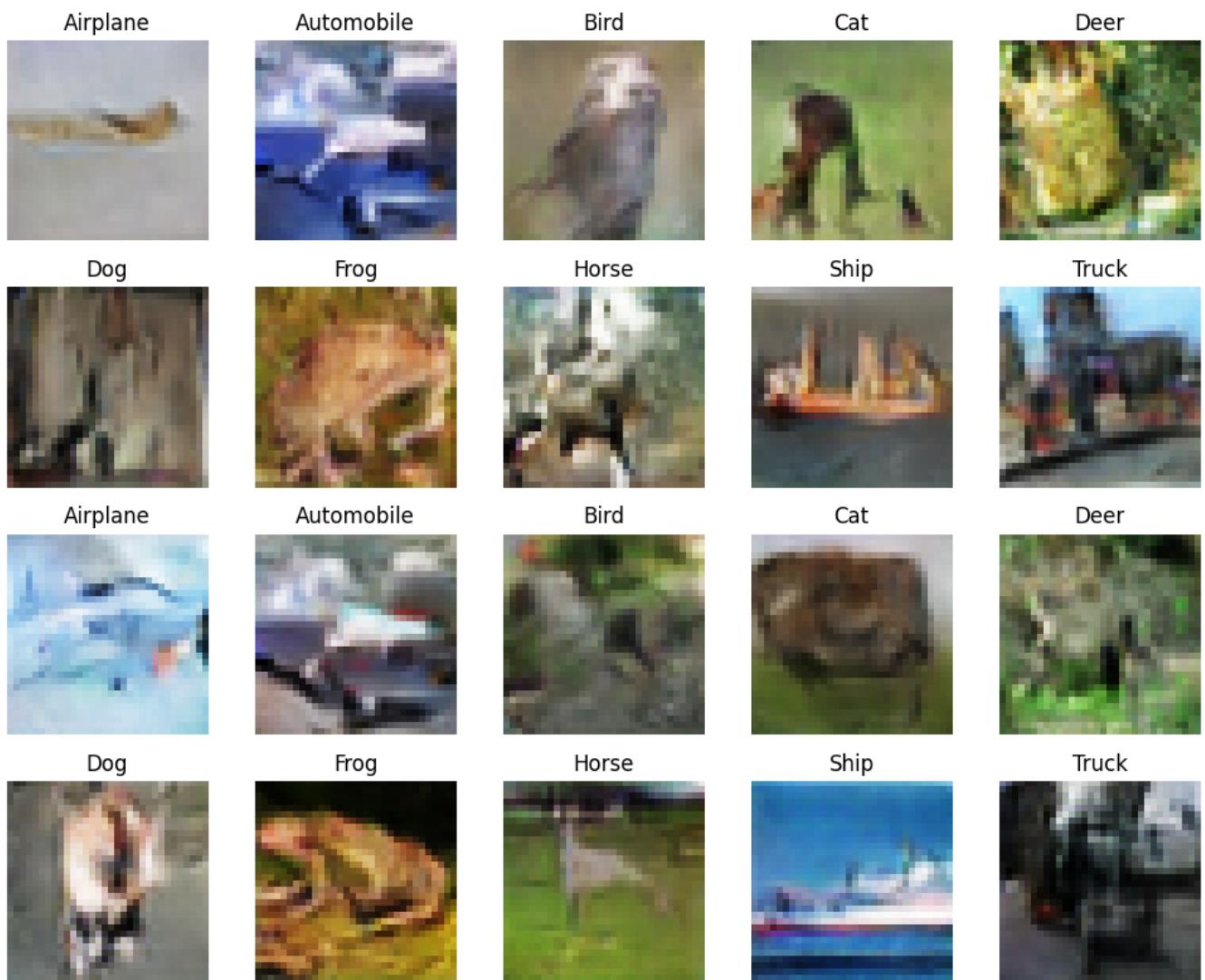
Epoch 179/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0692 - g_loss: 8.1587 - D(x|y): 0.4998 - D(G(z|y)): 0.0344 - KL Divergence: 4.7506

Epoch 180/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0675 - g_loss: 8.3777 - D(x|y): 0.4996 - D(G(z|y)): 0.0301 - KL Divergence: 4.4855

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New cGAN/generator-epoch-180.h5

Epoch 181/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0678 - g_loss: 8.5439 -
 $D(x|y)$: 0.5003 - $D(G(z|y))$: 0.0307 - KL Divergence: 4.7544

Epoch 182/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0722 - g_loss: 8.8189 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0297 - KL Divergence: 5.1058

Epoch 183/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0682 - g_loss: 8.4843 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0279 - KL Divergence: 4.8050

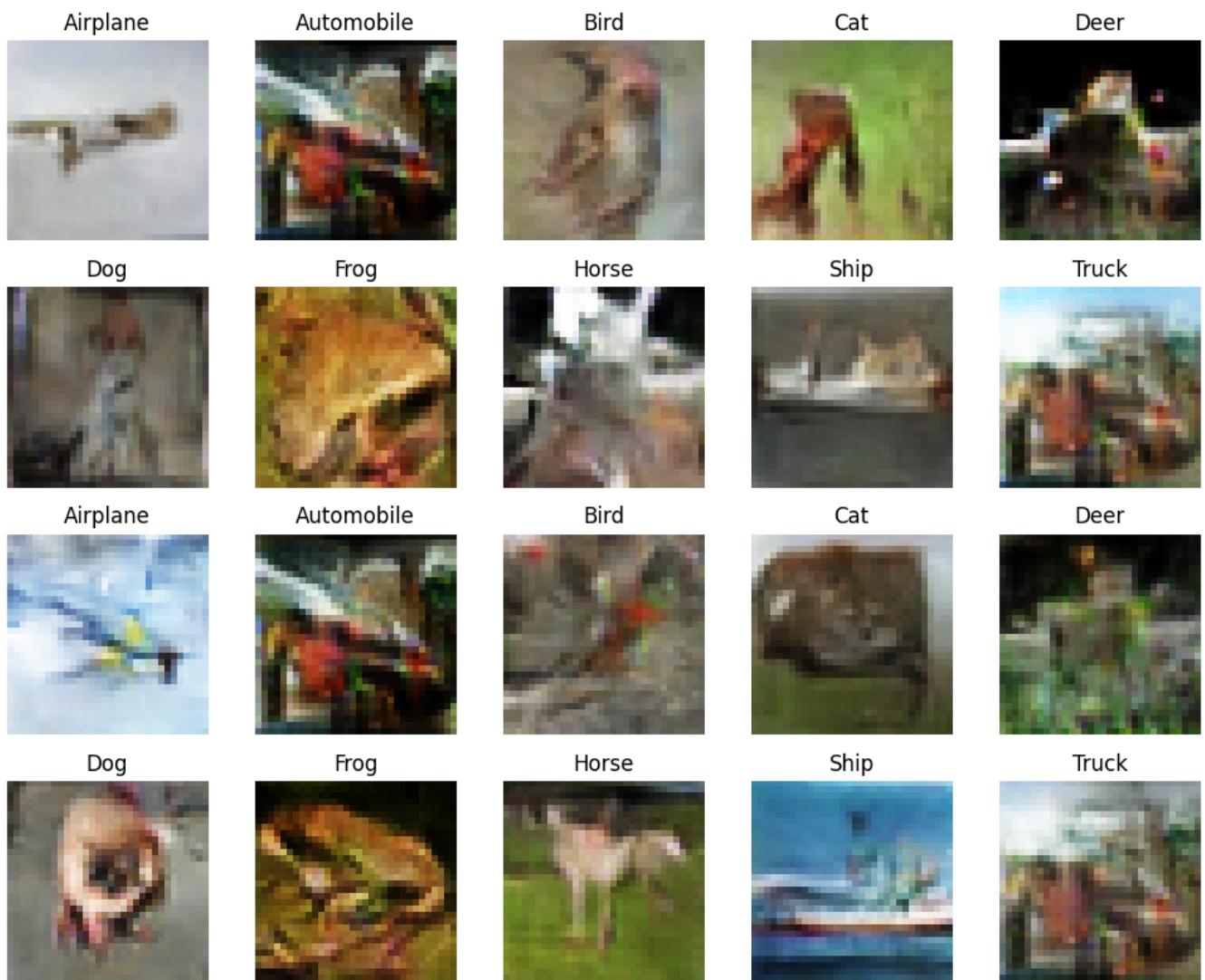
Epoch 184/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0744 - g_loss: 8.5693 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0318 - KL Divergence: 4.3674

Epoch 185/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0748 - g_loss: 8.4157 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0320 - KL Divergence: 4.7888

1/1 [=====] - 0s 28ms/step



Generator Checkpoint - New cGAN/generator-epoch-185.h5

Epoch 186/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0674 - g_loss: 8.5851 -
 $D(x|y)$: 0.4999 - $D(G(z|y))$: 0.0307 - KL Divergence: 4.8173

Epoch 187/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0689 - g_loss: 8.6554 -
 $D(x|y)$: 0.5006 - $D(G(z|y))$: 0.0301 - KL Divergence: 4.7703

Epoch 188/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0638 - g_loss: 8.2742 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0279 - KL Divergence: 4.6779

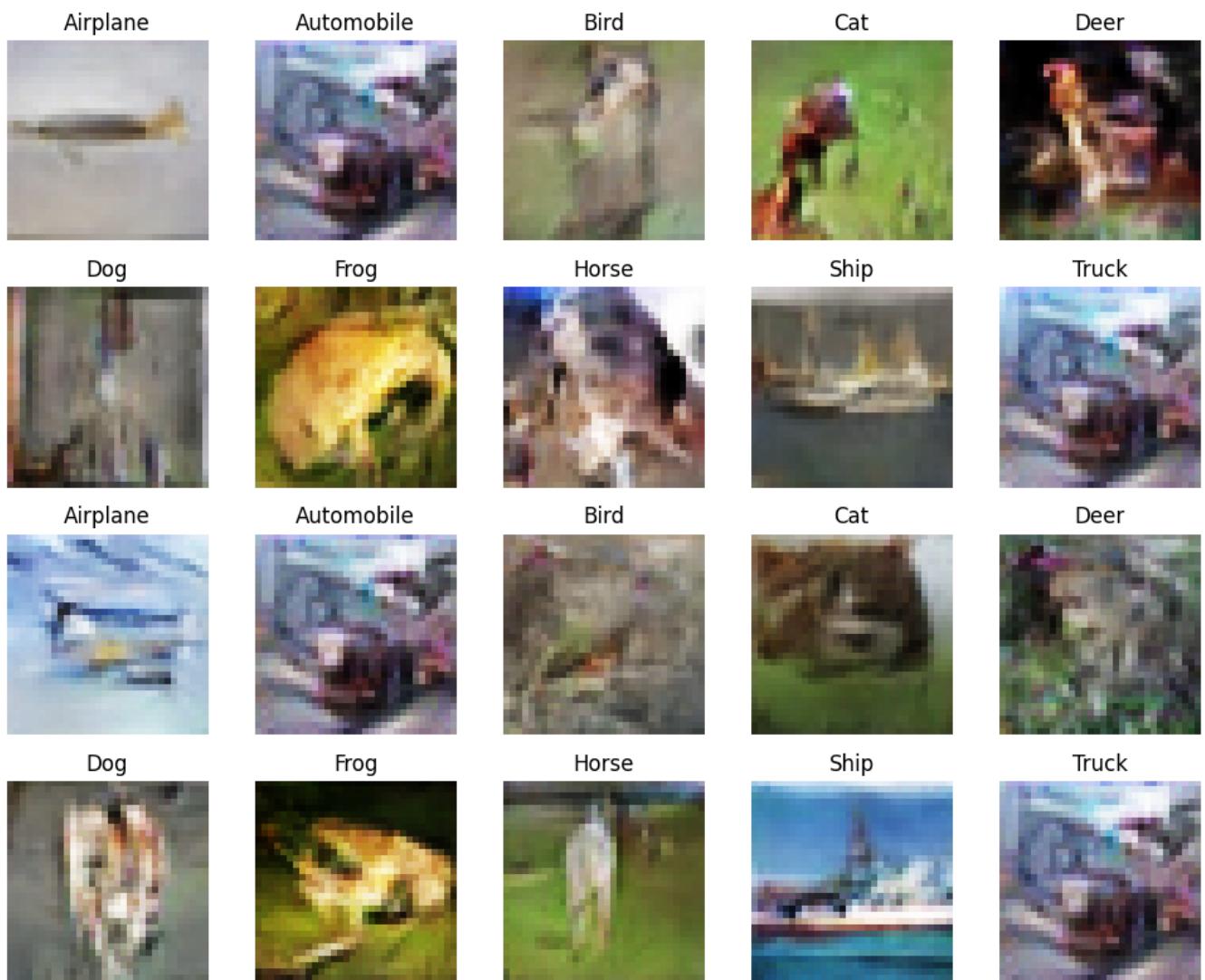
Epoch 189/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0690 - g_loss: 8.2845 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0300 - KL Divergence: 4.8092

Epoch 190/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0676 - g_loss: 8.6302 -
 $D(x|y)$: 0.5004 - $D(G(z|y))$: 0.0304 - KL Divergence: 4.5259

1/1 [=====] - 0s 26ms/step



Generator Checkpoint - New cGAN/generator-epoch-190.h5

Epoch 191/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0712 - g_loss: 8.2033 - D(x|y): 0.4999 - D(G(z|y)): 0.0330 - KL Divergence: 4.4581

Epoch 192/200

782/782 [=====] - 47s 60ms/step - d_loss: 0.0613 - g_loss: 8.3229 - D(x|y): 0.5003 - D(G(z|y)): 0.0280 - KL Divergence: 4.6061

Epoch 193/200

782/782 [=====] - 46s 59ms/step - d_loss: 0.0628 - g_loss: 8.5700 - D(x|y): 0.5003 - D(G(z|y)): 0.0294 - KL Divergence: 4.5541

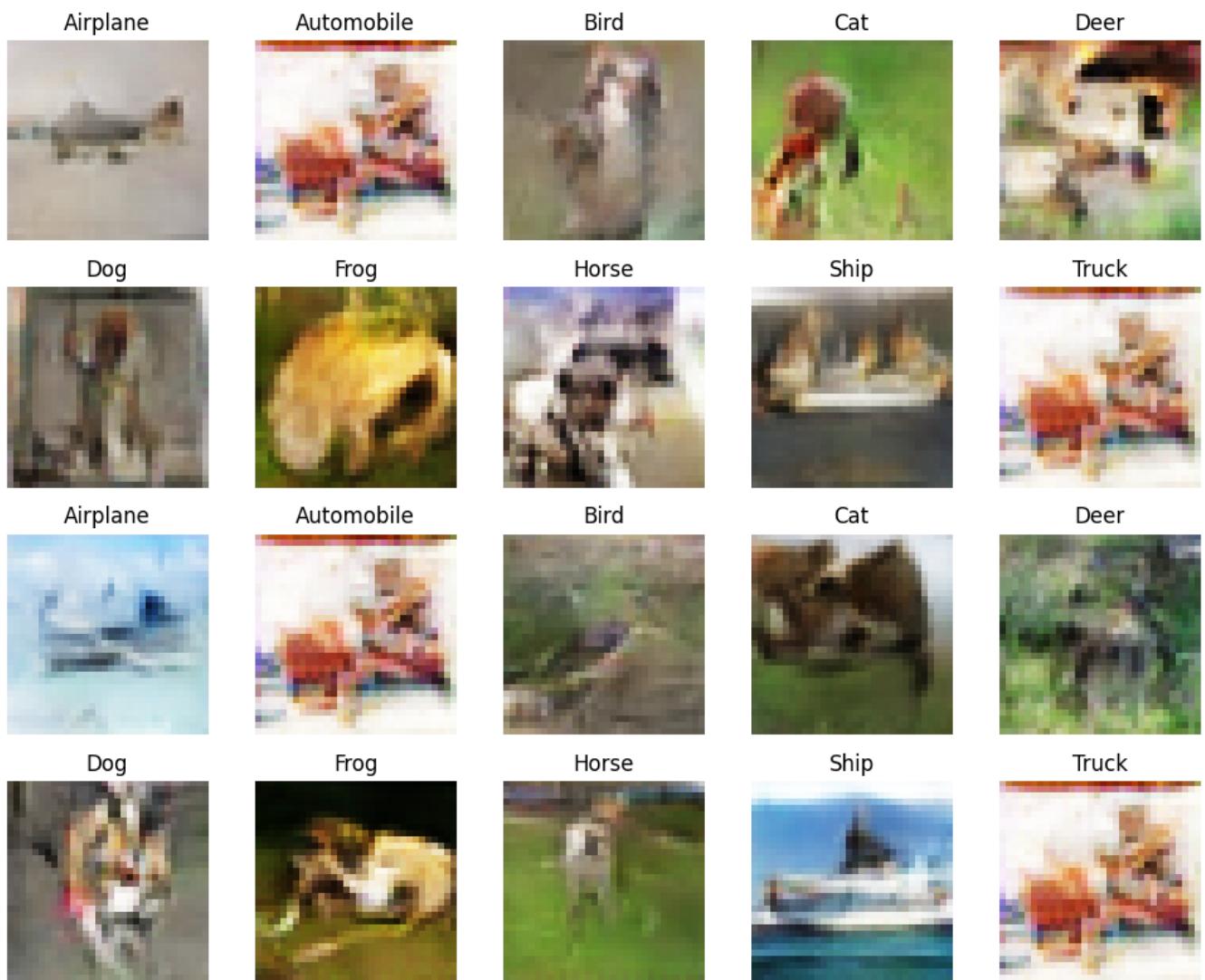
Epoch 194/200

782/782 [=====] - 45s 58ms/step - d_loss: 0.0654 - g_loss: 8.2692 - D(x|y): 0.5003 - D(G(z|y)): 0.0296 - KL Divergence: 4.7508

Epoch 195/200

782/782 [=====] - 44s 56ms/step - d_loss: 0.0648 - g_loss: 8.3486 - D(x|y): 0.5002 - D(G(z|y)): 0.0285 - KL Divergence: 4.2113

1/1 [=====] - 0s 24ms/step



Generator Checkpoint - New cGAN/generator-epoch-195.h5

Epoch 196/200

782/782 [=====] - 44s 57ms/step - d_loss: 0.0693 - g_loss: 8.6436 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0313 - KL Divergence: 4.1144

Epoch 197/200

782/782 [=====] - 44s 56ms/step - d_loss: 0.0653 - g_loss: 8.1464 -
 $D(x|y)$: 0.5001 - $D(G(z|y))$: 0.0312 - KL Divergence: 4.6433

Epoch 198/200

782/782 [=====] - 45s 57ms/step - d_loss: 0.0634 - g_loss: 8.2508 -
 $D(x|y)$: 0.5000 - $D(G(z|y))$: 0.0292 - KL Divergence: 4.7606

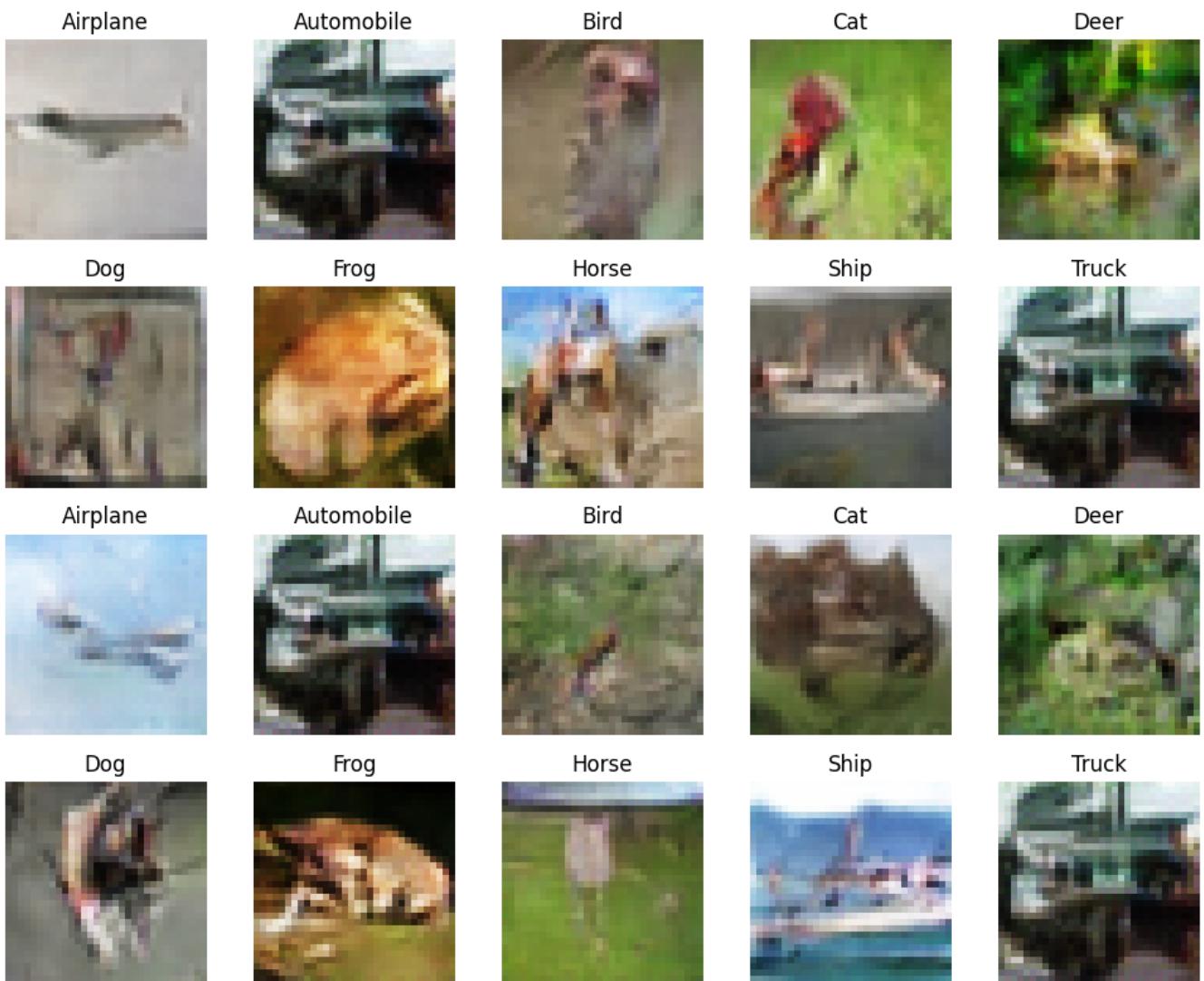
Epoch 199/200

782/782 [=====] - 44s 57ms/step - d_loss: 0.0652 - g_loss: 8.3266 -
 $D(x|y)$: 0.5002 - $D(G(z|y))$: 0.0283 - KL Divergence: 4.7572

Epoch 200/200

782/782 [=====] - 44s 57ms/step - d_loss: 0.0699 - g_loss: 8.2233 -
 $D(x|y)$: 0.5005 - $D(G(z|y))$: 0.0306 - KL Divergence: 4.0840

1/1 [=====] - 0s 25ms/step



Generator Checkpoint - New cGAN/generator-epoch-Full Train.h5

Observations

At around 180 - 185 epochs, we note that the models have collapsed as Truck and Automobile starts repeating the image that was generated. Around 190 epochs, the generator starts generating images of Truck and Automobile that is exactly the same. We note that other class looks generated looks good.

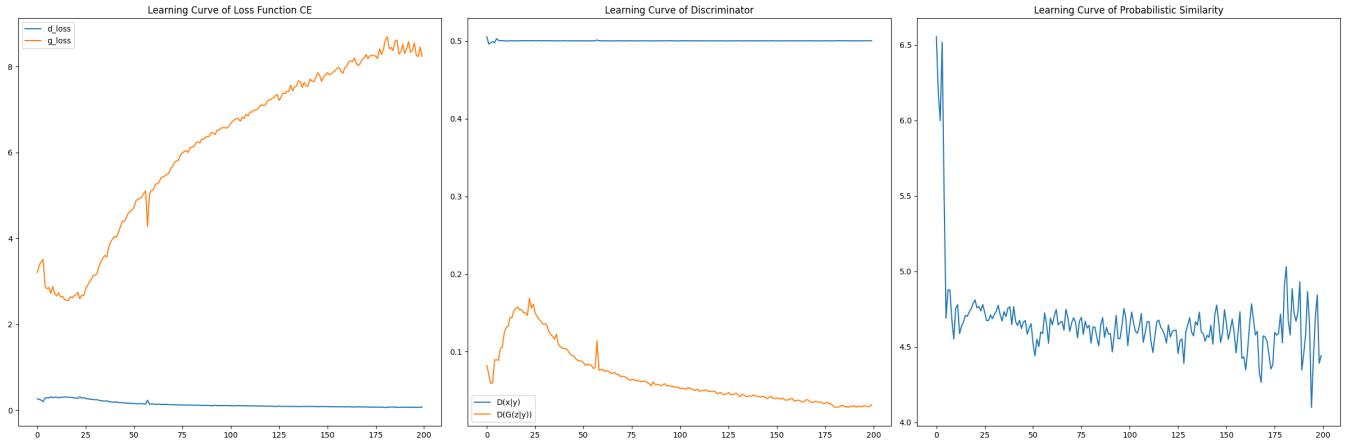
New cGAN Evaluation

As mentioned previously we will be using a quantitative and manual evaluation of the cGAN.

We will start off with a quantitative analysis so that we can find the best model to generate the images for manual evaluation.

```
In [ ]: # store history object into dataframe
improve_cond_gan_hist_df = pd.DataFrame(improve_cond_gan_hist.history)

# using pandas dataframe to plot out Learning curve
fig, (ax1, ax2, ax3) = plt.subplots(
    1, 3, figsize=(24, 8), tight_layout=True)
improve_cond_gan_hist_df.loc[:, ["d_loss", 'g_loss']].plot(
    ax=ax1, title=r'Learning Curve of Loss Function CE')
improve_cond_gan_hist_df.loc[:, ['D(x|y)', 'D(G(z|y))']].plot(
    ax=ax2, title=r'Learning Curve of Discriminator')
improve_cond_gan_hist_df.loc[:, 'KL Divergence'].plot(
    ax=ax3, title=r'Learning Curve of Probabilistic Similarity')
plt.show()
```



Observations

We see that KL divergence has more spikes in training as the generator steady decrease, this is likely due to the Gaussian Weight initialised making training slower. As 180 epochs the model collapsed, the lowest KL Divergence score is at 160 epochs.

```
In [ ]: # Loading Weights for best Generator
saved_weights = 'New cGAN\generator-epoch-160.h5'
improve_cond_gan.generator.load_weights(saved_weights)
improve_cond_gan.generator.summary()
```

Model: "generator_cGAN"

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
<hr/>			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_10 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_8 (Dense)	(None, 2048)	284672	['concatenate_10[0][0]']
batch_normalization_126 (Batch Normalization)	(None, 2048)	8192	['dense_8[0][0]']
p_re_lu_12 (PReLU)	(None, 2048)	2048	['batch_normalization_126[0][0]']
reshape_3 (Reshape)	(None, 2, 2, 512)	0	['p_re_lu_12[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2784320	['reshape_3[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
<hr/>			
=====			
Total params: 3,082,307			
Trainable params: 3,076,419			
Non-trainable params: 5,888			

In []:

```
n = 10000

# generating Labels
labels = np.random.randint(low=0, high=10, size=n)
one_hot_labels = to_categorical(labels)

# Generating 1000 Synthetic Images
random_noise = tf.random.normal(shape=(n, NOISE))

synthetic_images = improve_cond_gan.generator.predict(
    [random_noise, one_hot_labels])
print("Latent Vector Dim: {} \t Generated Images Dim: {}".format(
    random_noise.shape, synthetic_images.shape))

# Scaling back to [0, 1]
synthetic_images -= -1
synthetic_images /= (1 - (-1))

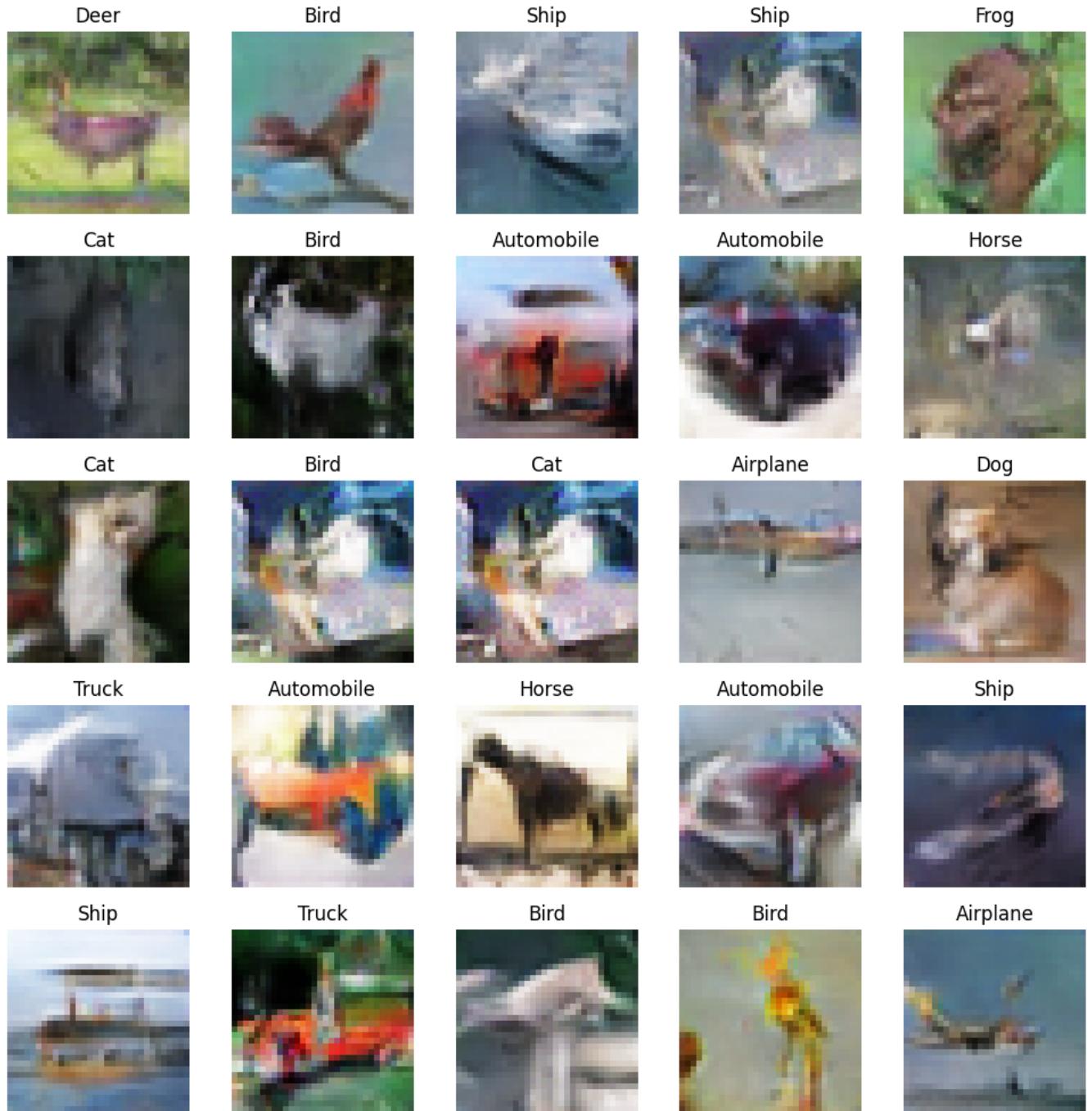
# Display 25 randomly sampled images
fig = plt.figure(figsize=(10, 10), tight_layout=True)
```

```

for i in range(25):
    rand_idx = np.random.randint(0, len(synthetic_images))
    ax = fig.add_subplot(5, 5, i+1)
    ax.imshow(synthetic_images[rand_idx])
    ax.set_title(class_labels[labels[rand_idx]])
    ax.axis('off')
plt.show()

```

313/313 [=====] - 3s 11ms/step
Latent Vector Dim: (10000, 128) Generated Images Dim: (10000, 32, 32, 3)



Observations

We note that the shape of the different classes can be seen clearly. However one issue is the images from other classes have generated the same images. This suggest that there might have been an issue with the generator when generating images.

```

In [ ]: improve_cond_gan_fid_class = GAN_FID(batch_size=512, noise=128,
                                             sample_size=10000, buffer_size=1024, labels=True)
improve_cond_gan_fid_class.fit(
    generator=improve_cond_gan.generator, train_data=x_train)
fid_score = improve_cond_gan_fid_class.evaluate()

```

```

0%|          | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 5s 124ms/step
16/16 [=====] - 2s 125ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 128ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 127ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 146ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 129ms/step
16/16 [=====] - 2s 130ms/step
Computing Generated Image Embeddings
0%|          | 0/20 [00:00<?, ?it/s]
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 130ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 132ms/step
16/16 [=====] - 2s 131ms/step
16/16 [=====] - 2s 135ms/step
16/16 [=====] - 2s 157ms/step
16/16 [=====] - 3s 158ms/step
16/16 [=====] - 3s 160ms/step
16/16 [=====] - 3s 161ms/step
16/16 [=====] - 3s 163ms/step
16/16 [=====] - 3s 163ms/step
16/16 [=====] - 3s 159ms/step
16/16 [=====] - 3s 158ms/step
16/16 [=====] - 3s 161ms/step
16/16 [=====] - 3s 165ms/step
16/16 [=====] - 3s 164ms/step
Computed Embeddings      Real Images Embedding Shape: (10240, 2048)      Generated Images Embe
dding Shape: (10240, 2048)
The computed FID score is: 93.59766081757371

```

Observations

We see the FID score is lower than the original cGAN model from 94.536161025068 to 93.59766081757371. This suggest that the changes made did improve the model. We also note that the images for certain classes are the same image which might have affected the FID score.

Saving Models

As we have the models save in the individual folders, we will also need to save it in the models folder. As we will be generating 1000 images, we will be using the 2 models that have the lowest FID score which is the ACGAN and new cGAN. We will also load the DCGAN to do a comparison.

DCGAN

```
In [ ]: gan.generator.save("models/DCGAN.h5")
gan.generator.summary()
```

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built.
`model.compile_metrics` will be empty until you train or evaluate the model.
Model: "generator_GAN"

Layer (type)	Output Shape	Param #
=====		
dense_18 (Dense)	(None, 2048)	264192
reshape_7 (Reshape)	(None, 2, 2, 512)	0
conv2d_transpose_22 (Conv2D Transpose)	(None, 4, 4, 256)	2097408
batch_normalization_165 (BatchNormalization)	(None, 4, 4, 256)	1024
p_re_lu_19 (PReLU)	(None, 4, 4, 256)	4096
conv2d_transpose_23 (Conv2D Transpose)	(None, 8, 8, 128)	524416
batch_normalization_166 (BatchNormalization)	(None, 8, 8, 128)	512
p_re_lu_20 (PReLU)	(None, 8, 8, 128)	8192
conv2d_transpose_24 (Conv2D Transpose)	(None, 16, 16, 64)	131136
batch_normalization_167 (BatchNormalization)	(None, 16, 16, 64)	256
p_re_lu_21 (PReLU)	(None, 16, 16, 64)	16384
conv2d_transpose_25 (Conv2D Transpose)	(None, 32, 32, 3)	3075
=====		
Total params:	3,050,691	
Trainable params:	3,049,795	
Non-trainable params:	896	

ACGAN

```
In [ ]: aux_cond_gan.generator.save("models/ACGAN.h5")
aux_cond_gan.generator.summary()
```

WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built.
`model.compile_metrics` will be empty until you train or evaluate the model.
Model: "generator_cGAN"

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_16 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_14 (Dense)	(None, 2048)	284672	['concatenate_16[0][0]']
batch_normalization_150 (Batch Normalization)	(None, 2048)	8192	['dense_14[0][0]']
leaky_re_lu_40 (LeakyReLU)	(None, 2048)	0	['batch_normalization_150[0][0]']
reshape_5 (Reshape)	(None, 2, 2, 512)	0	['leaky_re_lu_40[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2754752	['reshape_5[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
<hr/>			
=====			
Total params:	3,050,691		
Trainable params:	3,045,699		
Non-trainable params:	4,992		

New cGAN

```
In [ ]: improve_cond_gan.generator.save("models/New cGAN.h5")
improve_cond_gan.generator.summary()
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built.  
`model.compile_metrics` will be empty until you train or evaluate the model.  
Model: "generator_cGAN"
```

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_10 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_8 (Dense)	(None, 2048)	284672	['concatenate_10[0][0]']
batch_normalization_126 (Batch Normalization)	(None, 2048)	8192	['dense_8[0][0]']
p_re_lu_12 (PReLU)	(None, 2048)	2048	['batch_normalization_126[0][0]']
reshape_3 (Reshape)	(None, 2, 2, 512)	0	['p_re_lu_12[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2784320	['reshape_3[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
<hr/>			
=====			
Total params: 3,082,307			
Trainable params: 3,076,419			
Non-trainable params: 5,888			

Generating 1000 images

As part of the project objective, we will be generating 1000 images. To help display the variety, we will use a .gif to show 100 images at a time [100 per class = 1000].

Utilities function

To create a gif to show, we need to use the HTML function to display the gif using image src.

```
In [ ]: def display_gif(gif):  
    return HTML(''.format(gif))
```

DCGAN

```
In [ ]: # Loading Generator from DCGAN  
gan_generator = tf.keras.models.load_model("./models/DCGAN.h5")  
gan_generator.summary()
```

WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.

Model: "generator_GAN"

Layer (type)	Output Shape	Param #
=====		
dense_18 (Dense)	(None, 2048)	264192
reshape_7 (Reshape)	(None, 2, 2, 512)	0
conv2d_transpose_22 (Conv2D Transpose)	(None, 4, 4, 256)	2097408
batch_normalization_165 (BatchNormalization)	(None, 4, 4, 256)	1024
p_re_lu_19 (PReLU)	(None, 4, 4, 256)	4096

Model: "generator_GAN"

Layer (type)	Output Shape	Param #
=====		
dense_18 (Dense)	(None, 2048)	264192
reshape_7 (Reshape)	(None, 2, 2, 512)	0
conv2d_transpose_22 (Conv2D Transpose)	(None, 4, 4, 256)	2097408
batch_normalization_165 (BatchNormalization)	(None, 4, 4, 256)	1024
p_re_lu_19 (PReLU)	(None, 4, 4, 256)	4096
conv2d_transpose_23 (Conv2D Transpose)	(None, 8, 8, 128)	524416
batch_normalization_166 (BatchNormalization)	(None, 8, 8, 128)	512
p_re_lu_20 (PReLU)	(None, 8, 8, 128)	8192
conv2d_transpose_24 (Conv2D Transpose)	(None, 16, 16, 64)	131136
batch_normalization_167 (BatchNormalization)	(None, 16, 16, 64)	256
p_re_lu_21 (PReLU)	(None, 16, 16, 64)	16384
conv2d_transpose_25 (Conv2D Transpose)	(None, 32, 32, 3)	3075
=====		
Total params:	3,050,691	
Trainable params:	3,049,795	
Non-trainable params:	896	

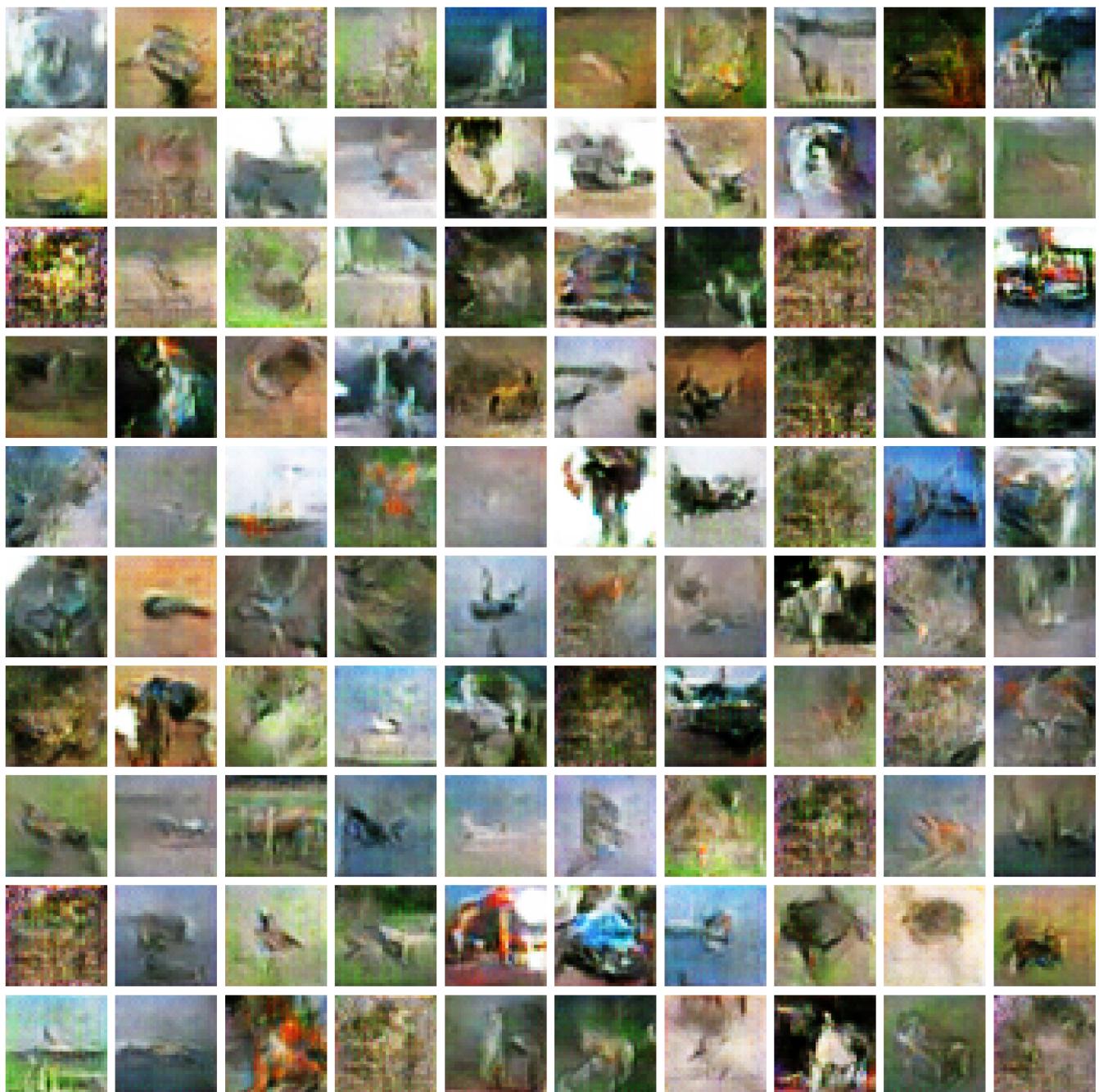
```
In [ ]: for i in range(10):
    # creating noise
    latent_z = np.random.normal(size=(100, 128))

    imgs = gan_generator.predict(latent_z)

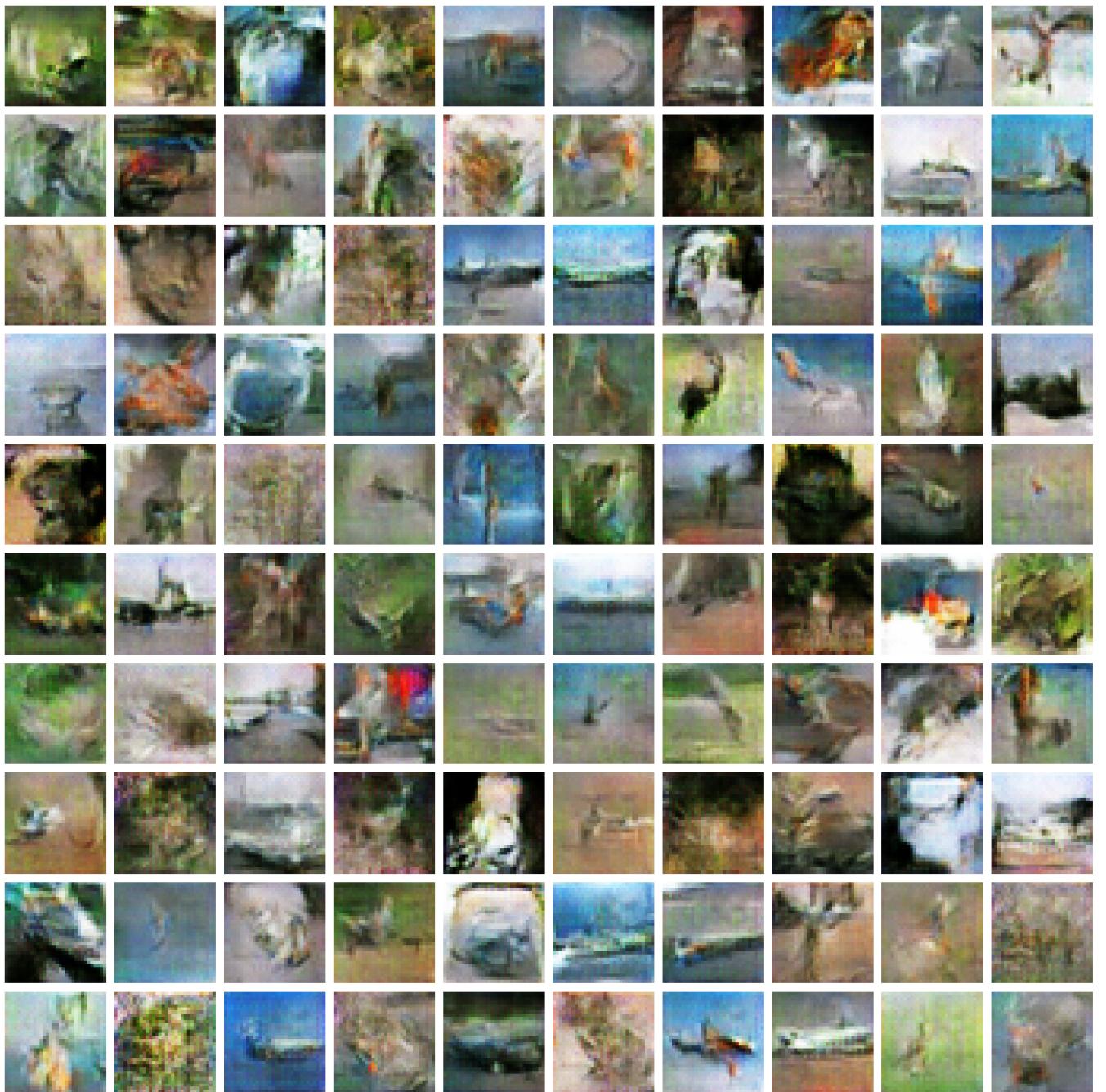
    fig = plt.figure(figsize=(20, 20), tight_layout=True)
```

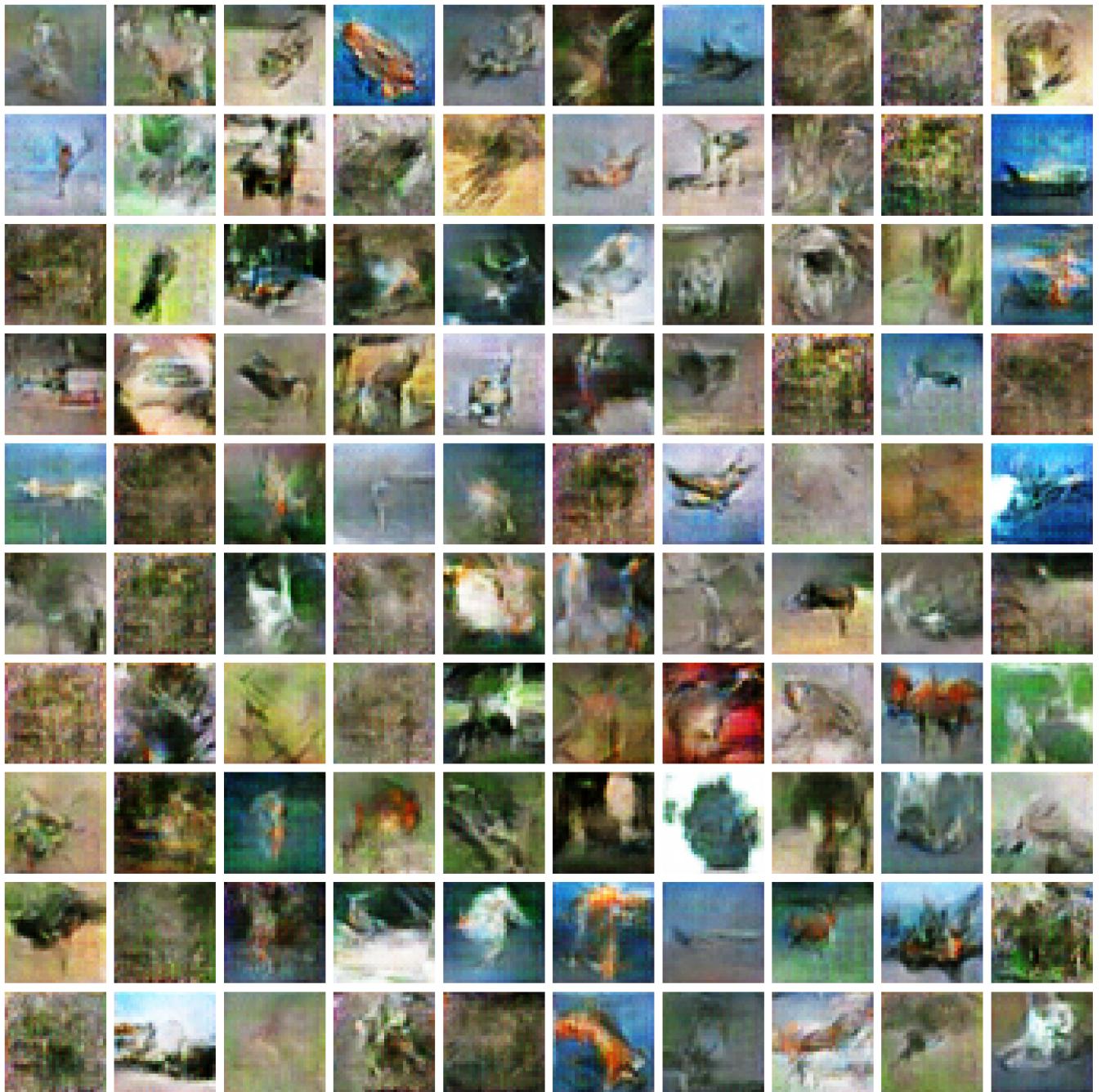
```
for idx, img in enumerate(imgs):
    ax = fig.add_subplot(10, 10, idx+1)
    ax.imshow((img + 1) / 2)
    ax.axis('off')
fig.savefig(fname='./images/GAN/{}.jpeg'.format(i))
```

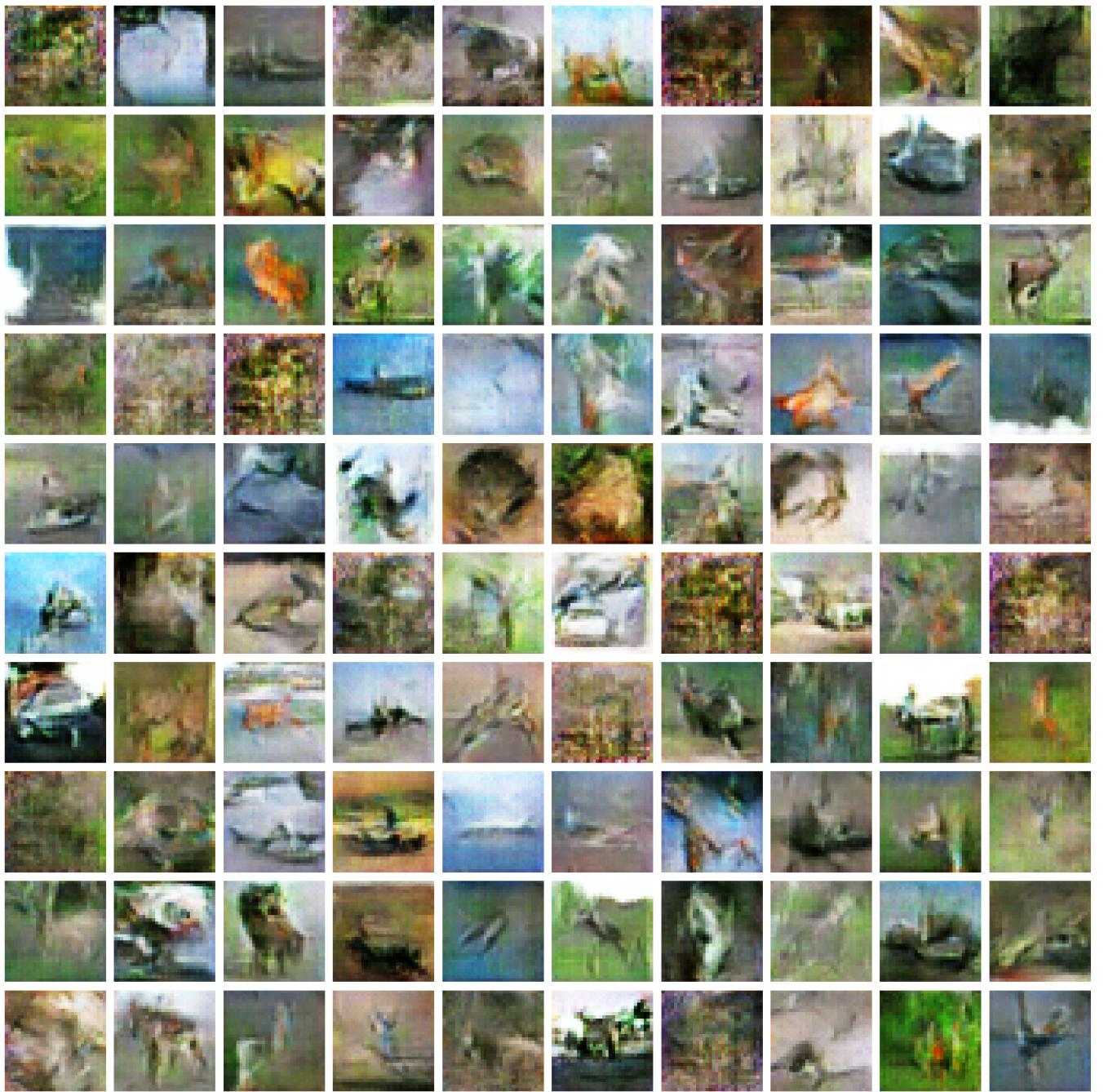
```
4/4 [=====] - 1s 32ms/step
4/4 [=====] - 0s 8ms/step
4/4 [=====] - 0s 8ms/step
4/4 [=====] - 0s 10ms/step
4/4 [=====] - 0s 8ms/step
4/4 [=====] - 0s 8ms/step
4/4 [=====] - 0s 9ms/step
4/4 [=====] - 0s 9ms/step
4/4 [=====] - 0s 8ms/step
4/4 [=====] - 0s 8ms/step
```

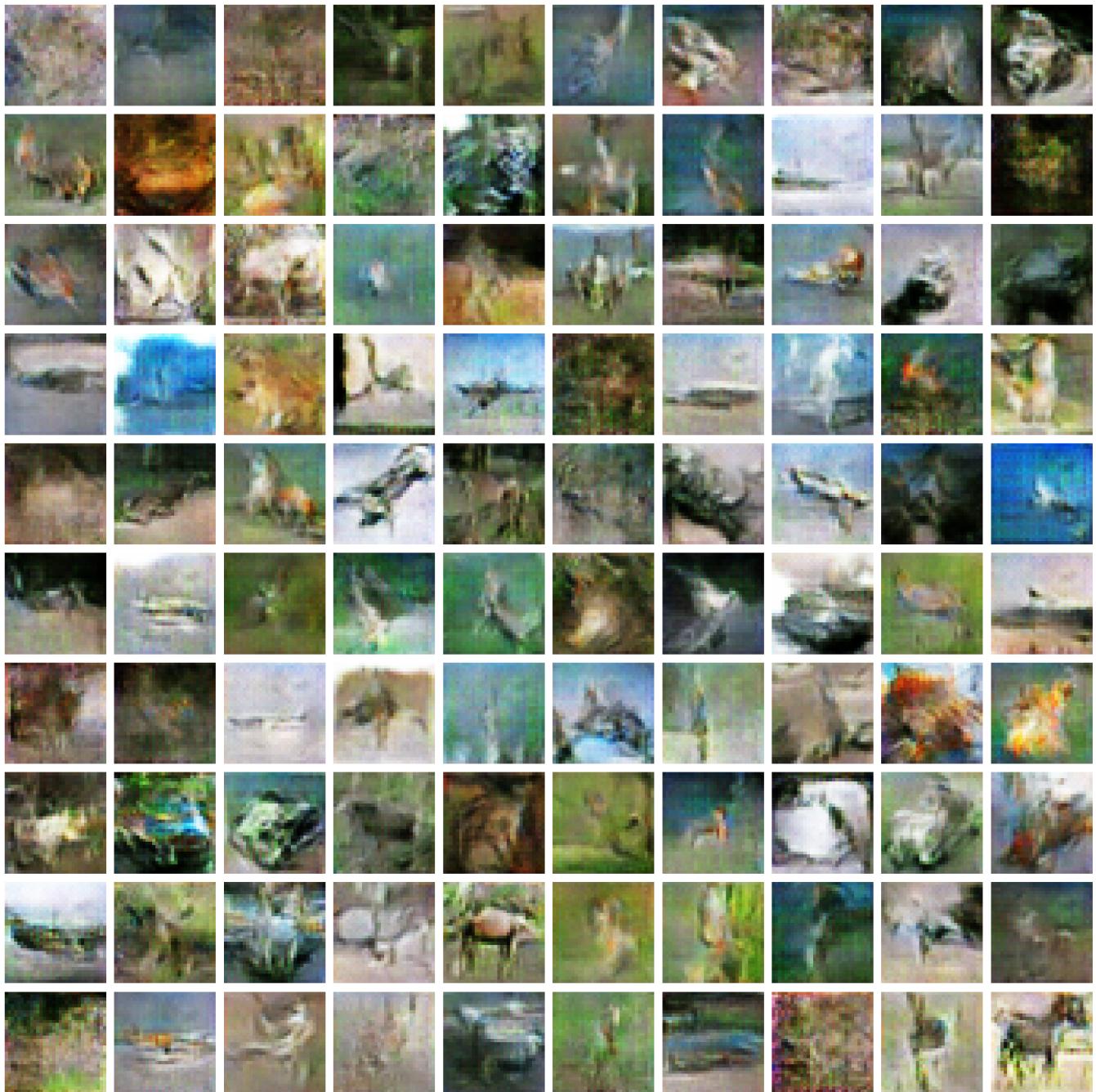




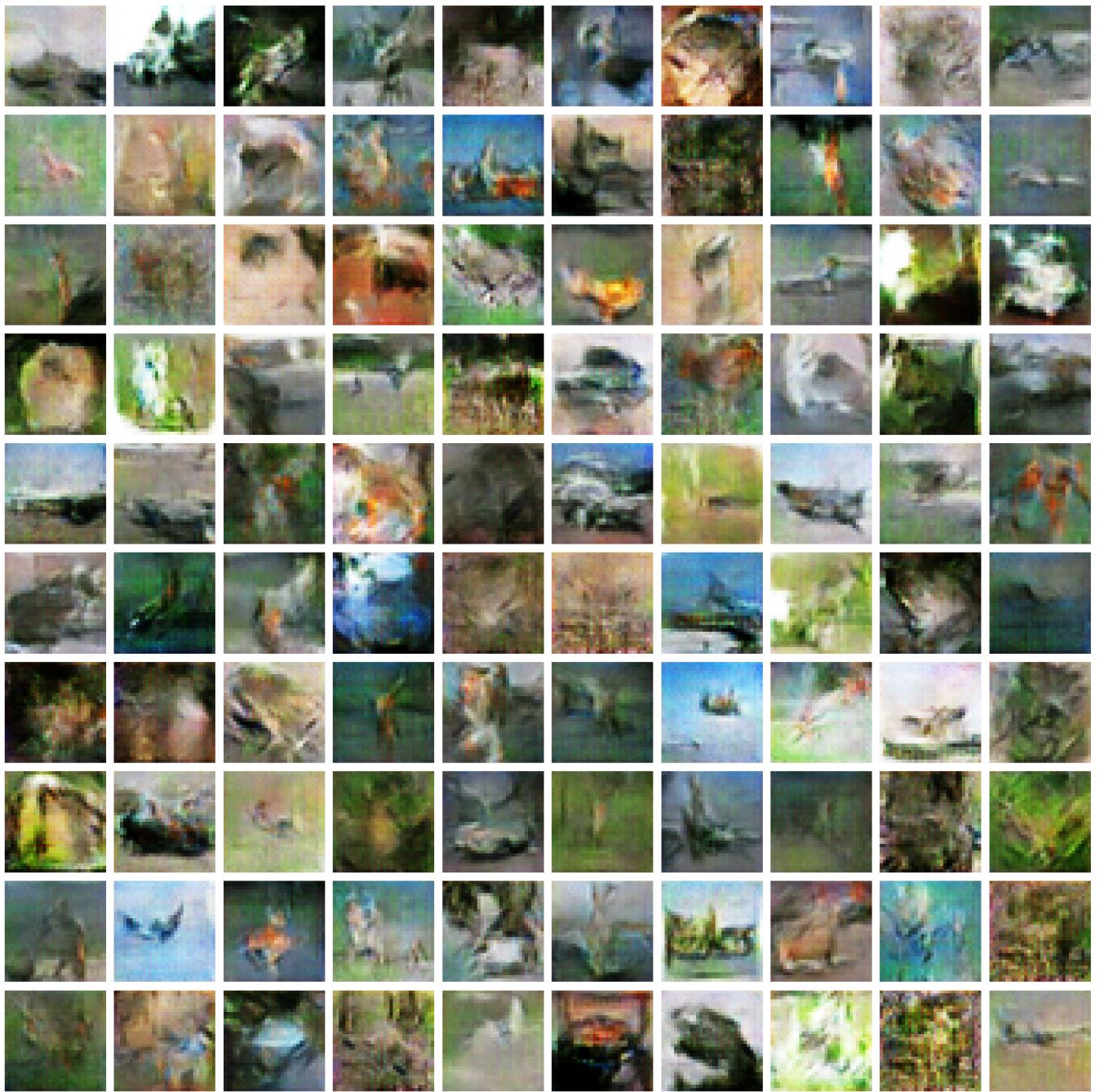


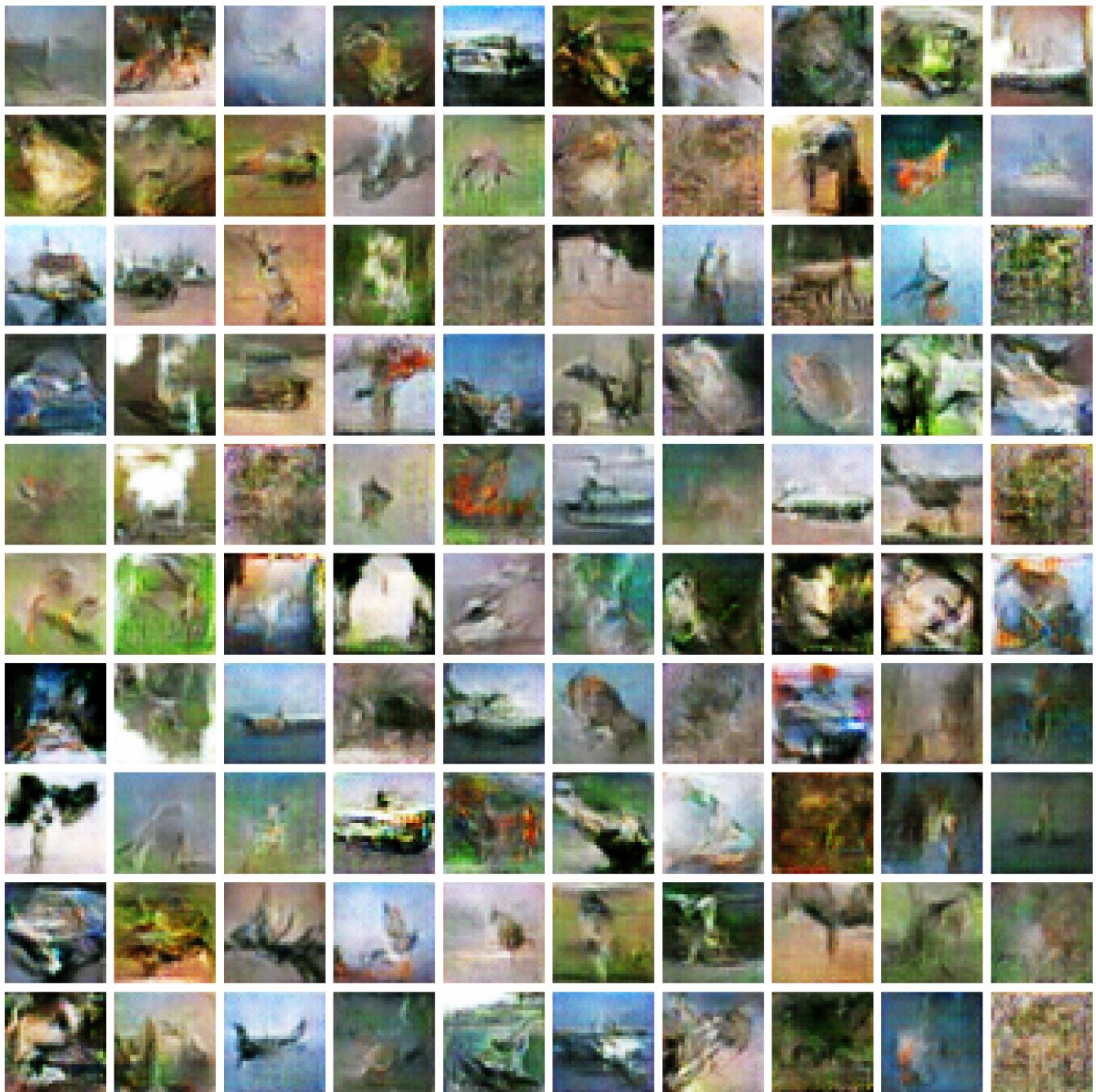


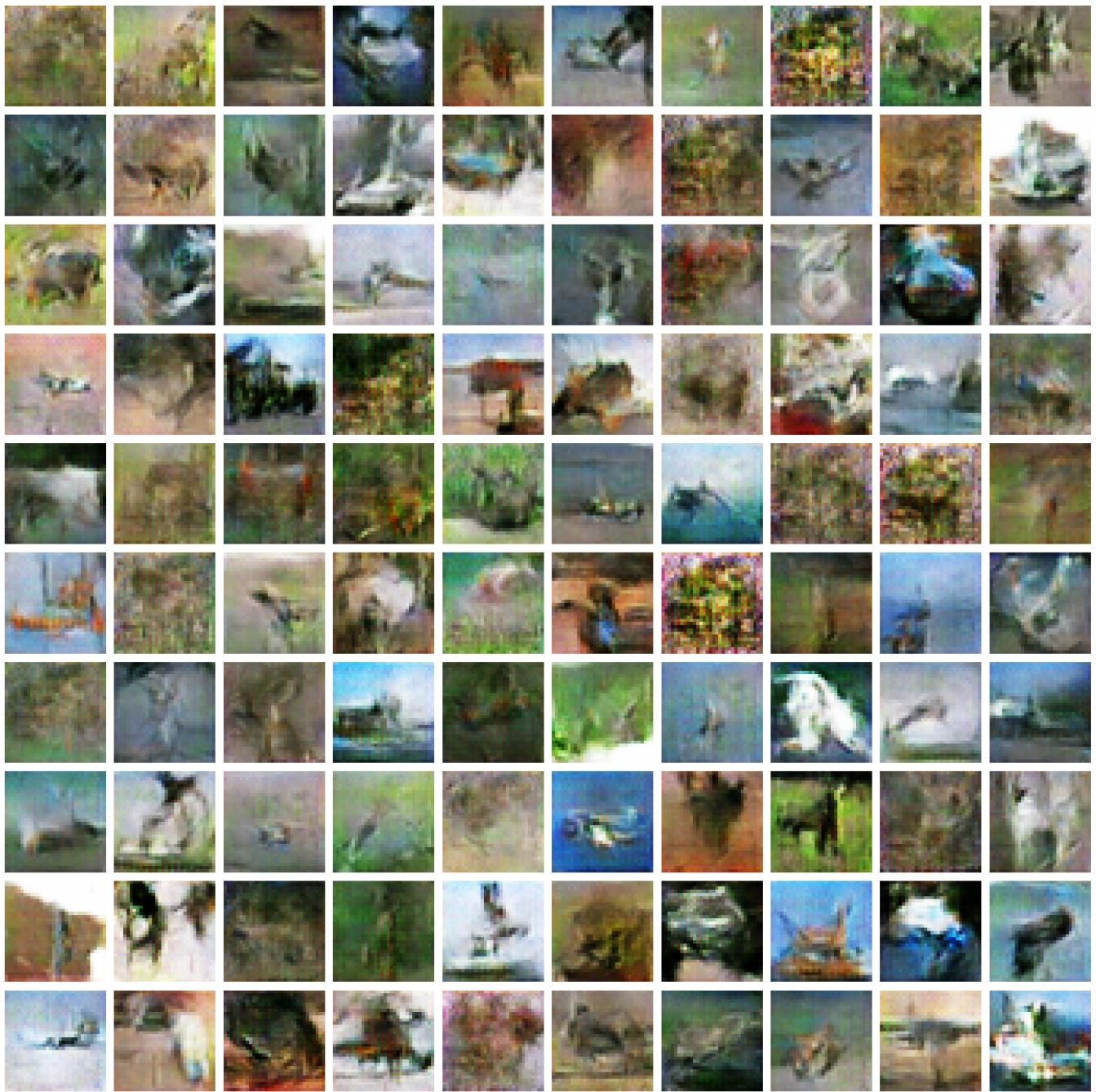










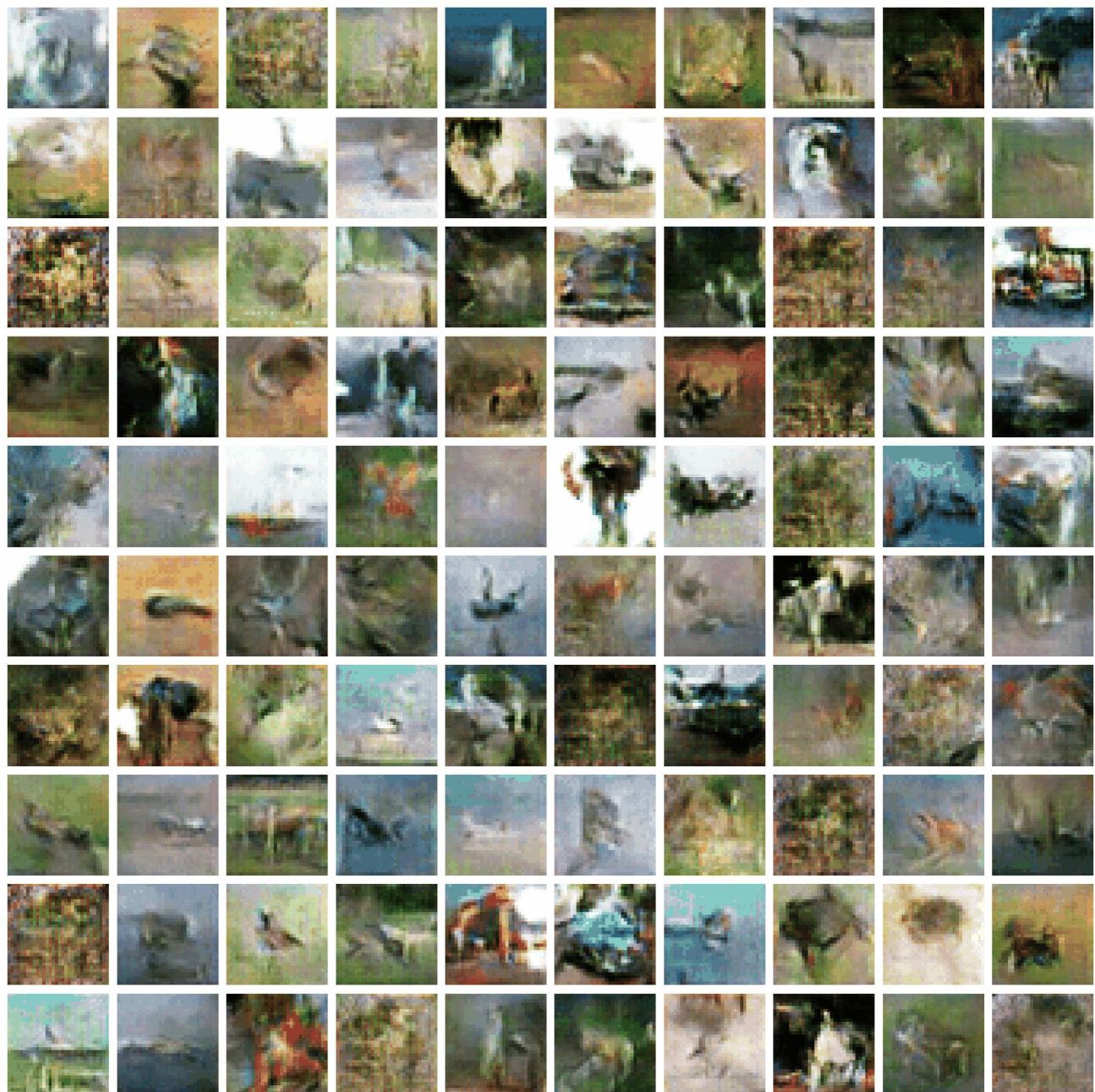


```
In [ ]: images = []
gan_img_paths = glob.glob("./images/GAN/*.jpeg")
for path in gan_img_paths:
    images.append(imageio.imread(path))
imageio.mimsave('./images/GAN/GAN.gif', images, duration=1)
display_gif('./images/GAN/GAN.gif')
```

C:\Users\Soh Hong Yu\AppData\Local\Temp\ipykernel_14940\1410452988.py:4: DeprecationWarning:
Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread.
To keep the current behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.

```
    images.append(imageio.imread(path))
```

Out[]:



ACGAN

In []:

```
# Loading Generator from ACGAN
aux_cond_gan_generator = tf.keras.models.load_model("./models/ACGAN.h5")
aux_cond_gan_generator.summary()
```

WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
Model: "generator_cGAN"

Layer (type)	Output Shape	Param #	Connected to
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_16 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0]', 'Conditions_y[0][0]']
dense_14 (Dense)	(None, 2048)	284672	['concatenate_16[0][0]']
batch_normalization_150 (Batch Normalization)	(None, 2048)	8192	['dense_14[0][0]']
Model: "generator_cGAN"			

Layer (type)	Output Shape	Param #	Connected to
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[None, 10]	0	[]
concatenate_16 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0]', 'Conditions_y[0][0]']
dense_14 (Dense)	(None, 2048)	284672	['concatenate_16[0][0]']
batch_normalization_150 (Batch Normalization)	(None, 2048)	8192	['dense_14[0][0]']
leaky_re_lu_40 (LeakyReLU)	(None, 2048)	0	['batch_normalization_150[0][0]']
reshape_5 (Reshape)	(None, 2, 2, 512)	0	['leaky_re_lu_40[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2754752	['reshape_5[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
=====			
Total params:	3,050,691		
Trainable params:	3,045,699		
Non-trainable params:	4,992		

```
In [ ]: for i in range(10):
    # creating conditions and noise
    conditions = np.full(100, i)
    conditions = tf.keras.utils.to_categorical(conditions, 10)
    latent_z = np.random.normal(size=(100, 128))

    imgs = aux_cond_gan_generator.predict([latent_z, conditions])
```

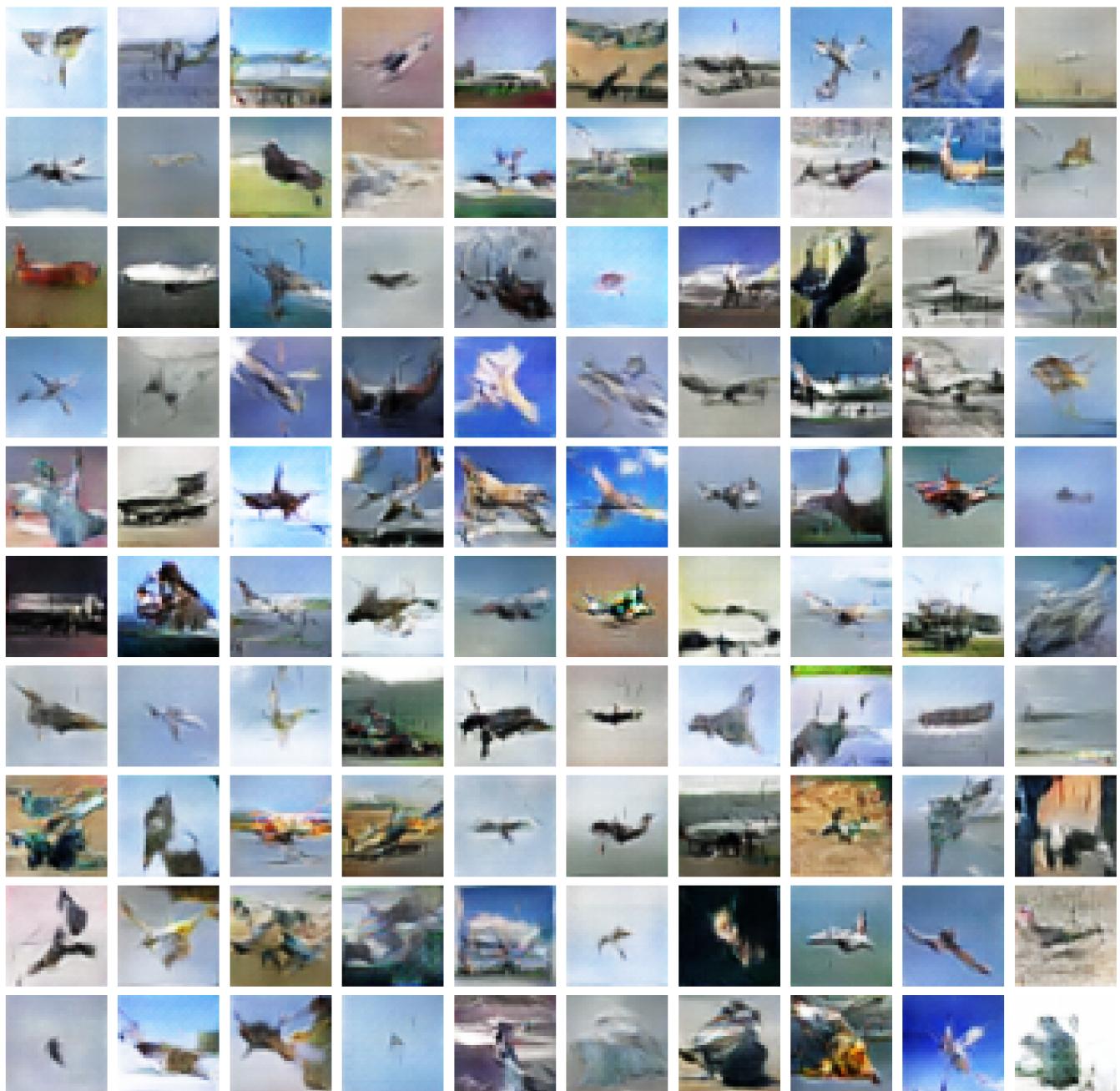
```

fig = plt.figure(figsize=(20, 20), tight_layout=True)
for idx, img in enumerate(imgs):
    ax = fig.add_subplot(10, 10, idx+1)
    ax.imshow((img + 1) / 2)
    ax.axis('off')
fig.suptitle(list(class_labels.values())[i], y=1, fontsize=28)
fig.savefig(fname="./images/ACGAN/{}.jpeg".format(list(class_labels.values())[i]))

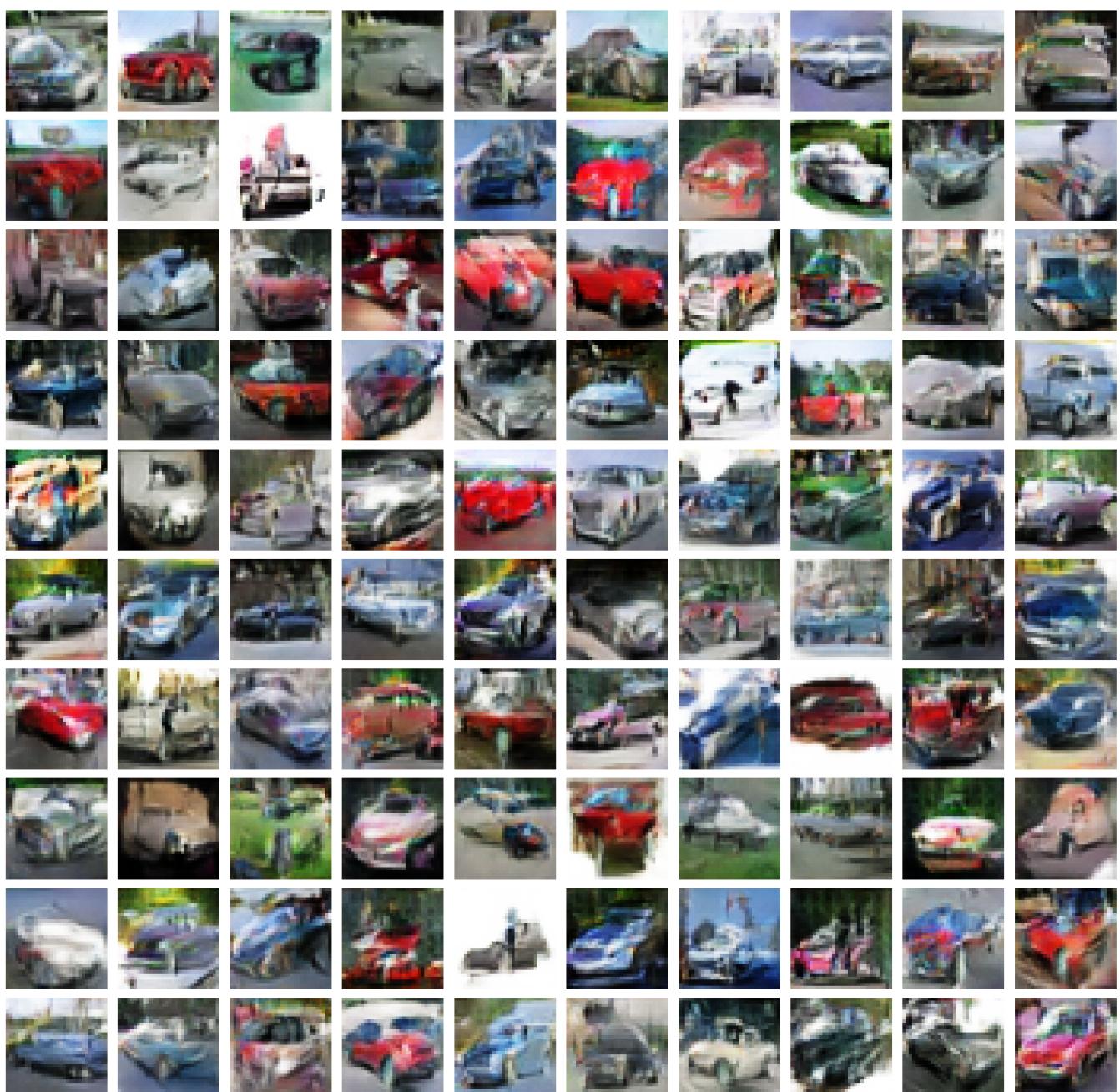
```

4/4 [=====] - 0s 9ms/step
 4/4 [=====] - 0s 8ms/step
 4/4 [=====] - 0s 6ms/step
 4/4 [=====] - 0s 6ms/step
 4/4 [=====] - 0s 7ms/step
 4/4 [=====] - 0s 11ms/step
 4/4 [=====] - 0s 8ms/step
 4/4 [=====] - 0s 8ms/step
 4/4 [=====] - 0s 7ms/step
 4/4 [=====] - 0s 12ms/step

Airplane



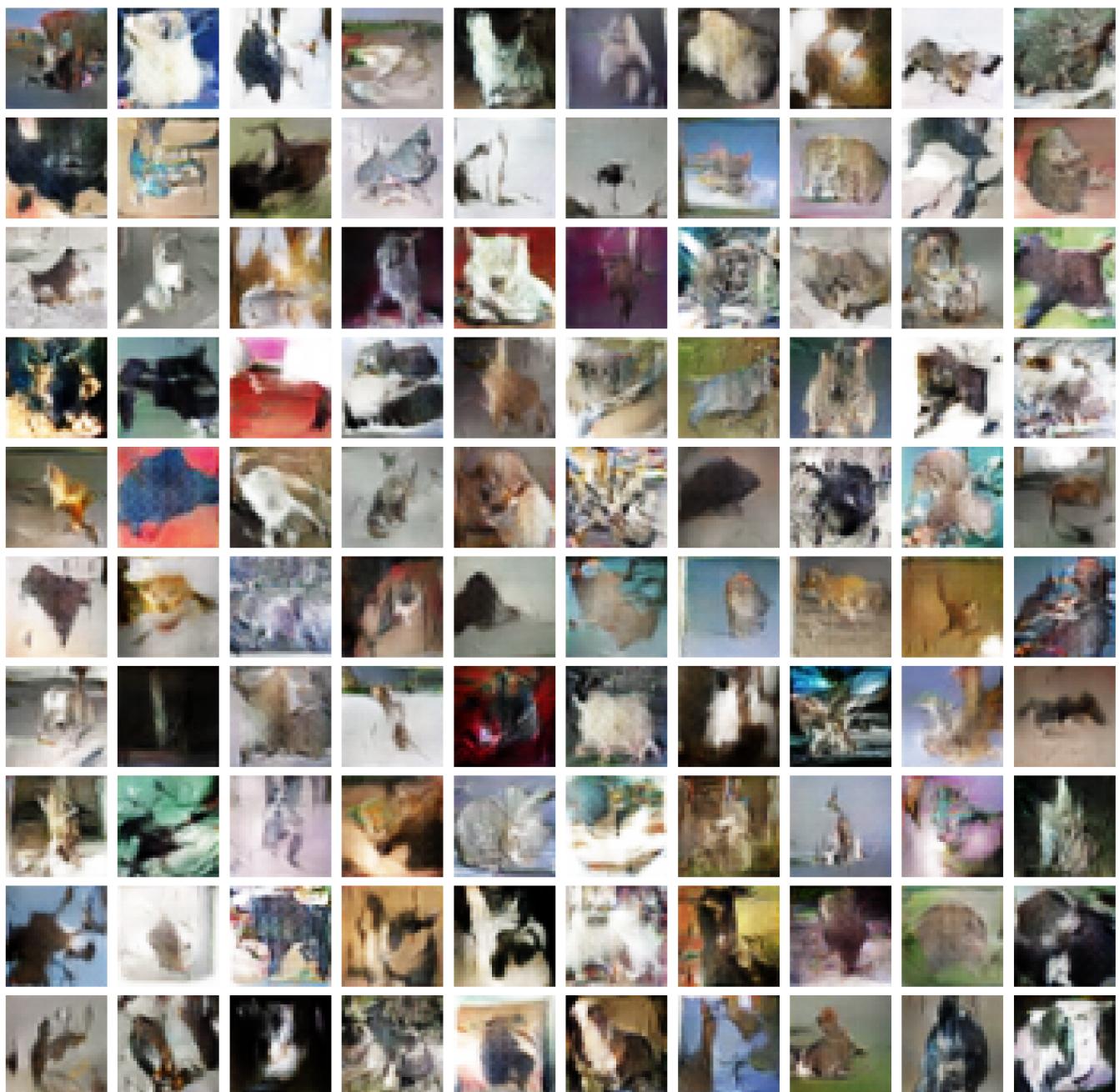
Automobile



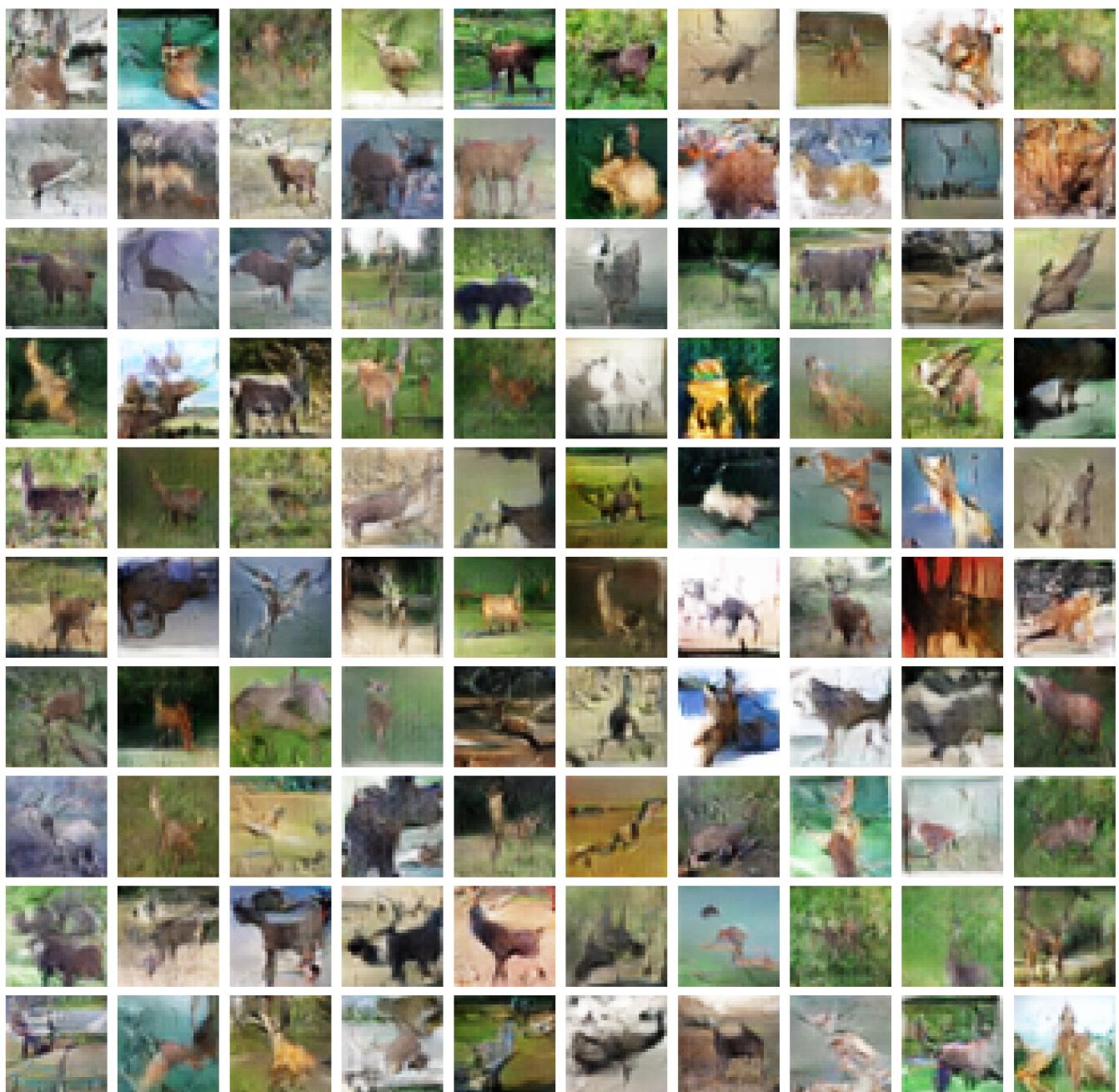
Bird



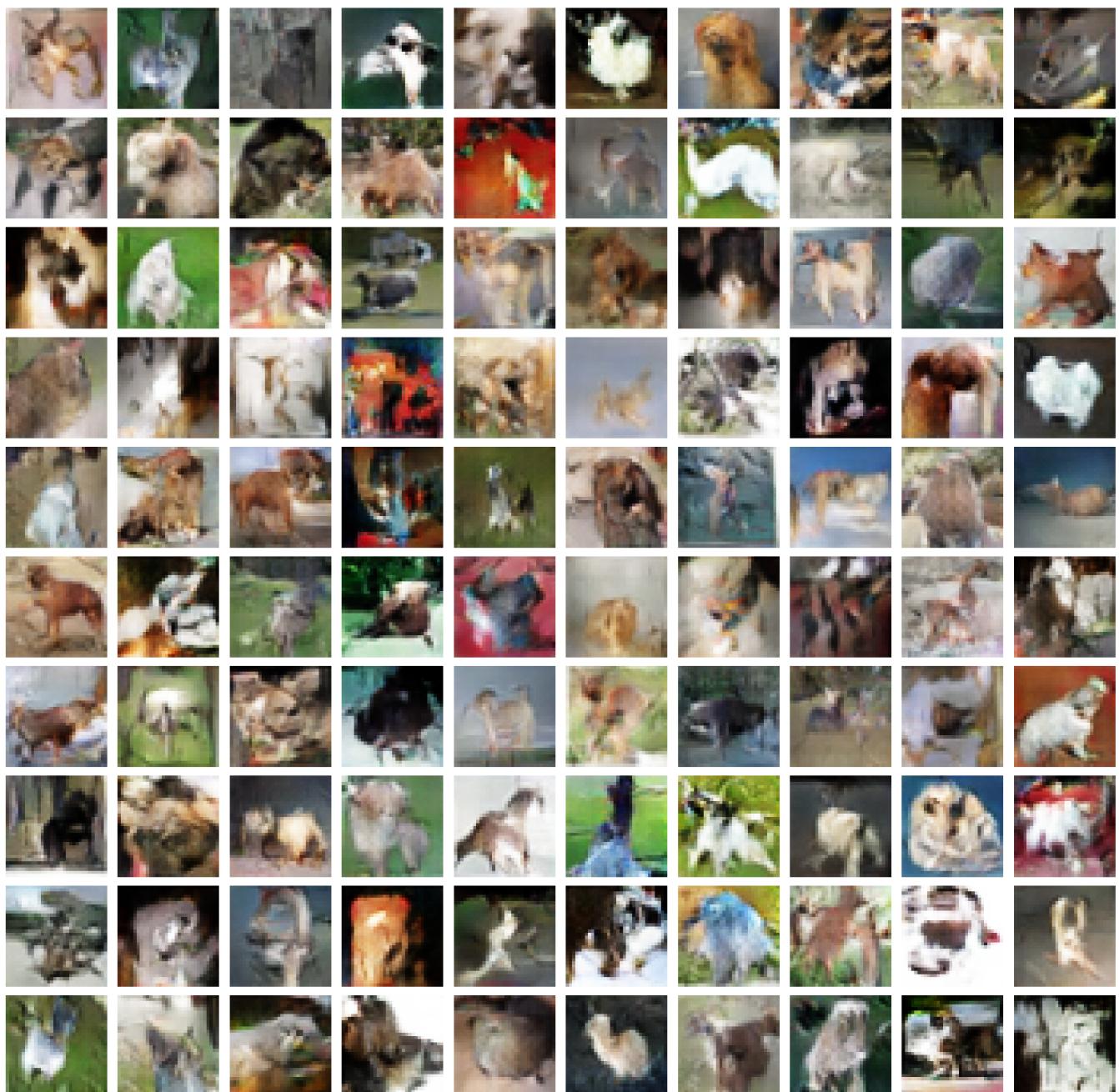
Cat



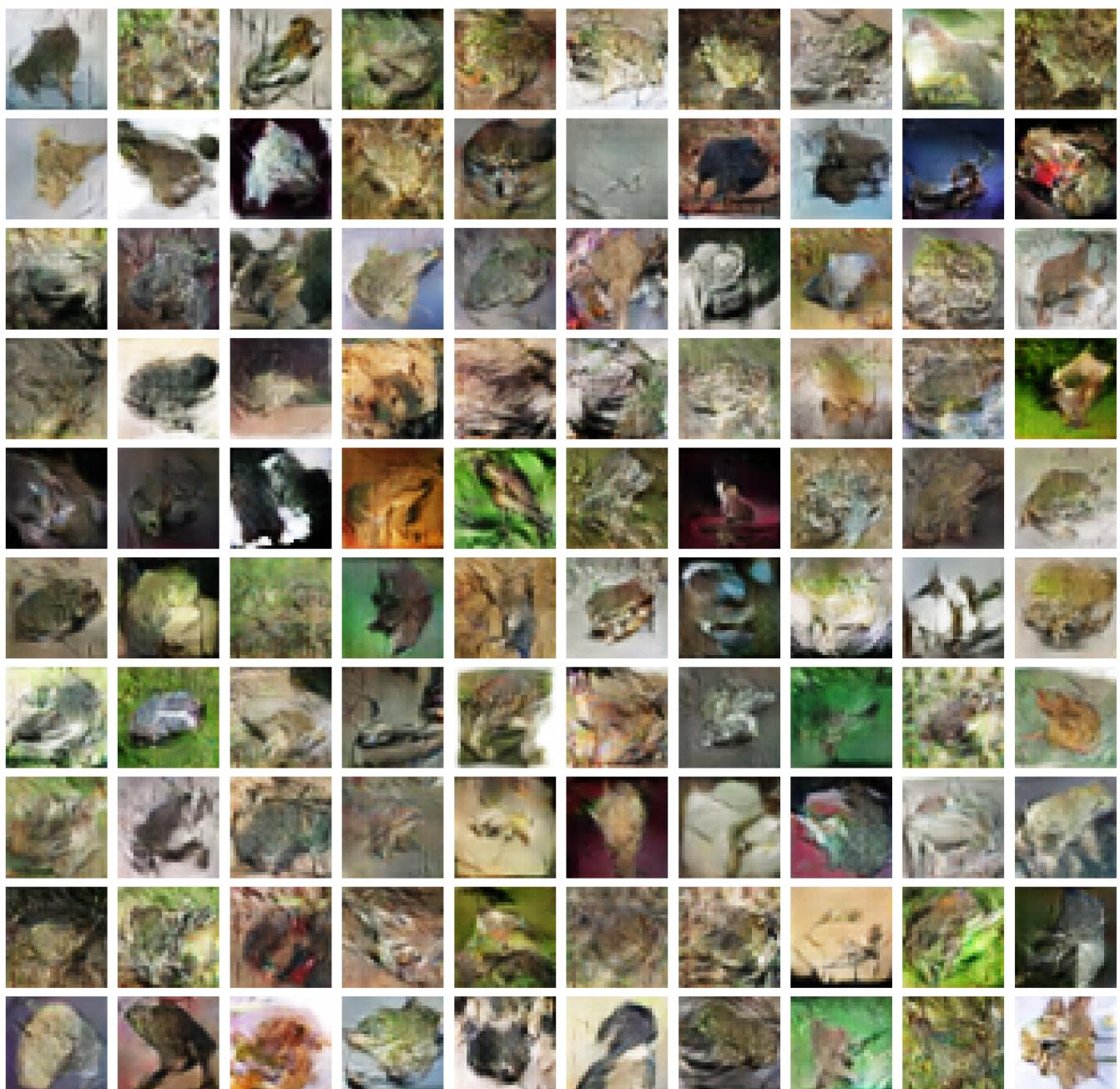
Deer



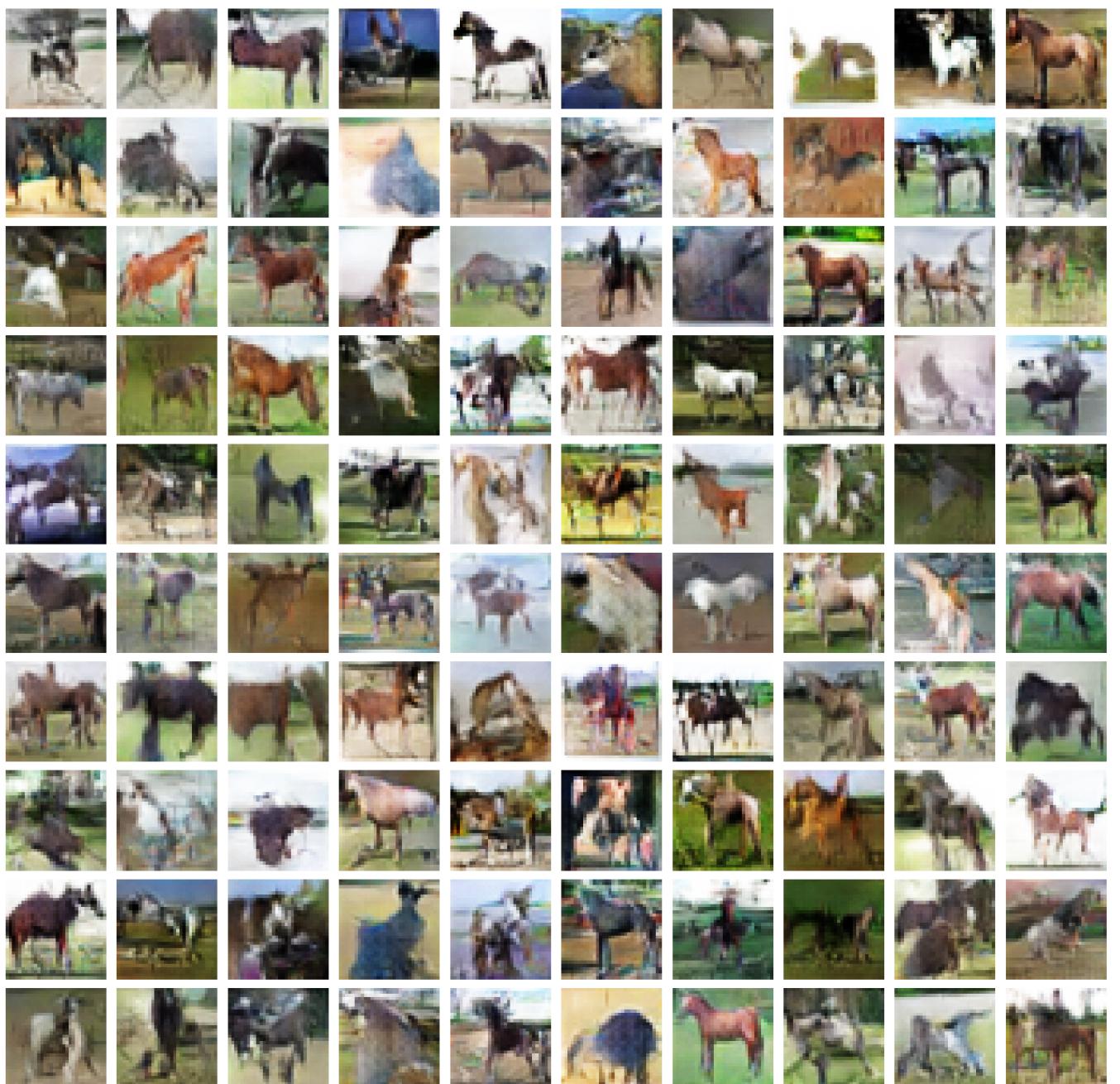
Dog



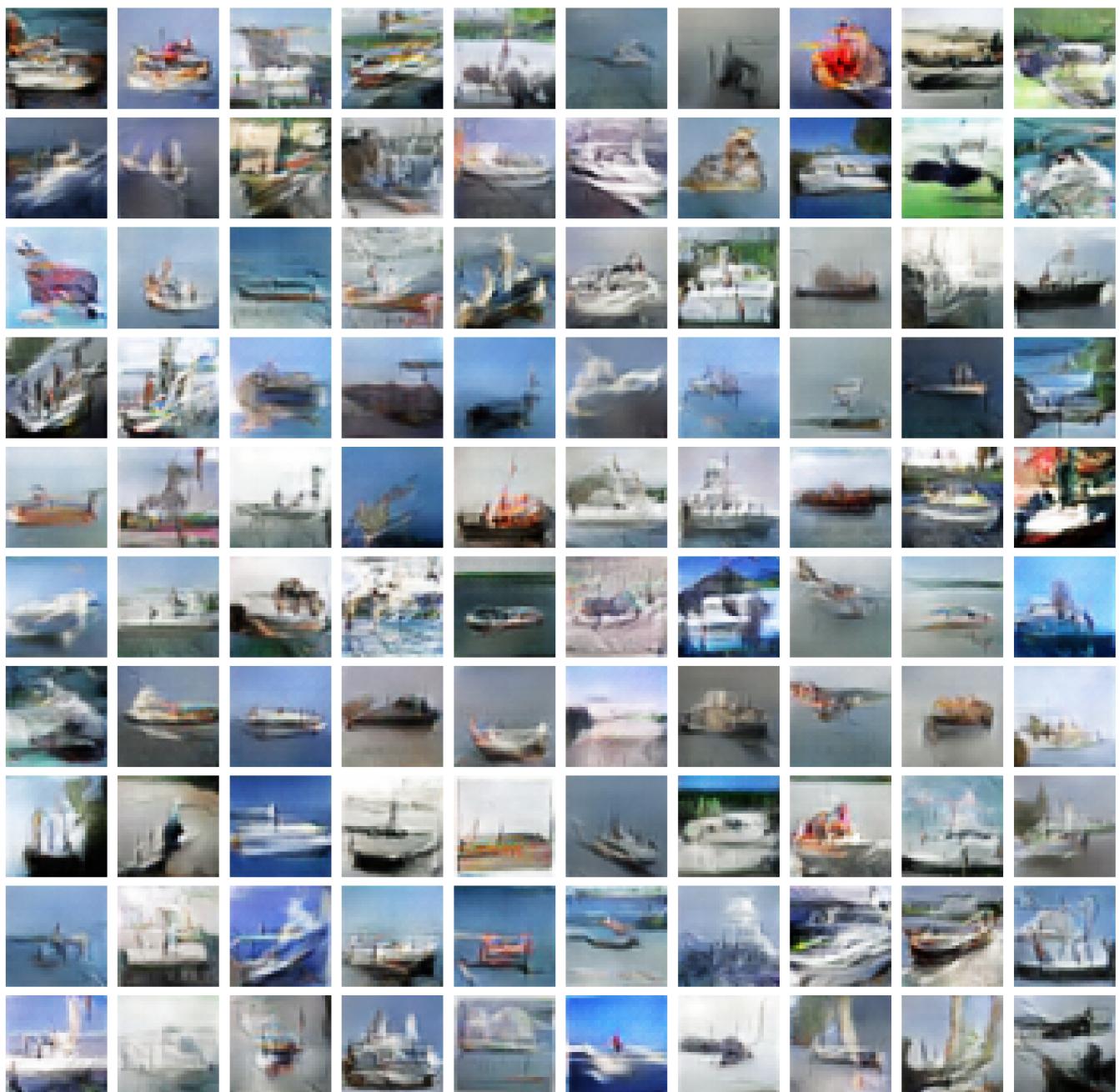
Frog



Horse



Ship



Truck

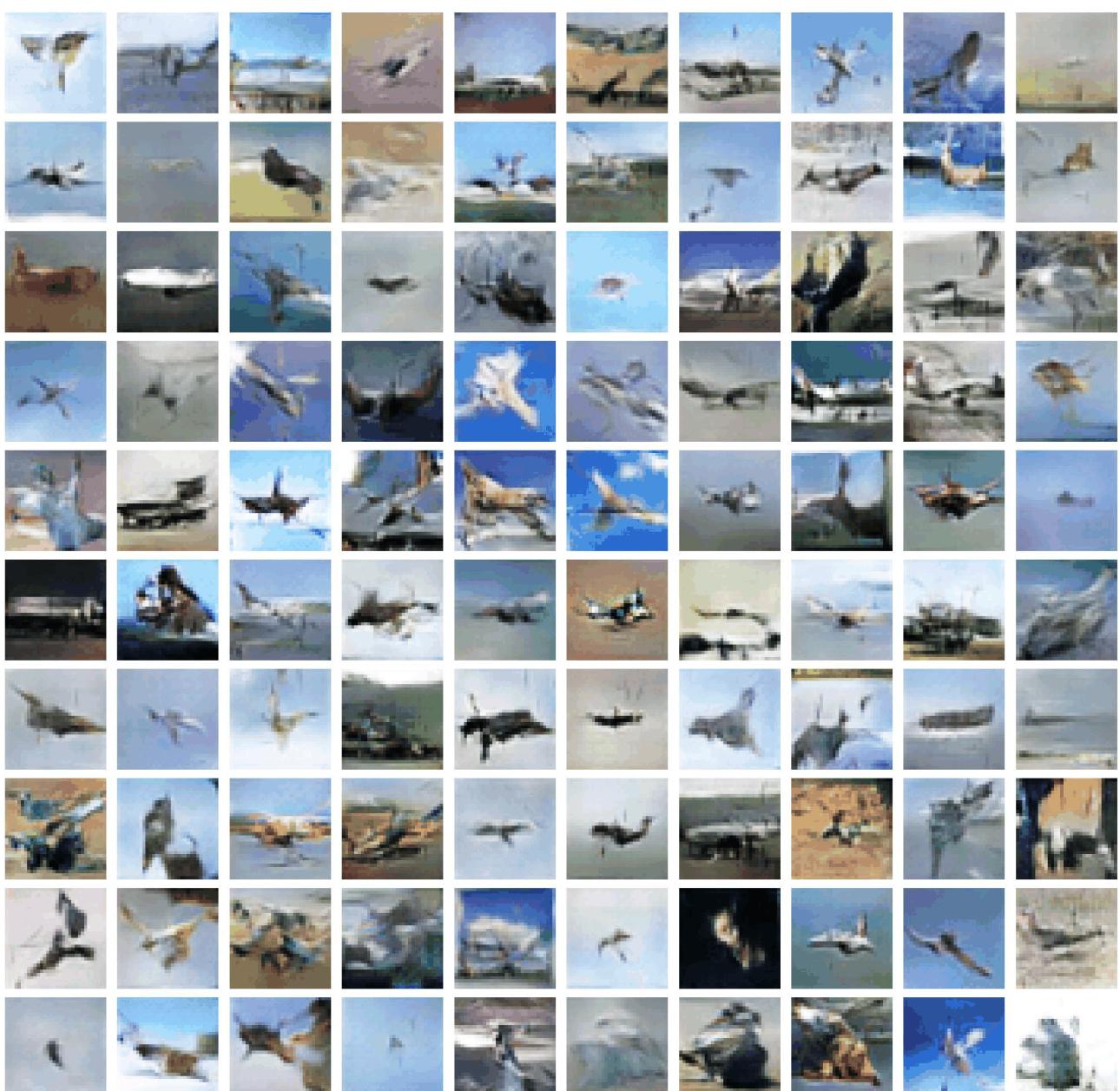


```
In [ ]: images = []
gan_img_paths = glob.glob("./images/ACGAN/*.jpeg")
for path in gan_img_paths:
    images.append(imageio.imread(path))
imageio.mimsave('./images/ACGAN/ACGAN.gif', images, duration=1)
display_gif('./images/ACGAN/ACGAN.gif')
```

C:\Users\Soh Hong Yu\AppData\Local\Temp\ipykernel_14940\899859476.py:4: DeprecationWarning: Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread. To keep the current behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.

```
    images.append(imageio.imread(path))
```

Out[]:



New cGAN

In []:

```
# Loading Generator from New cGAN
cond_gan_generator = tf.keras.models.load_model("./models/New cGAN.h5")
cond_gan_generator.summary()
```

```
c:\Users\Soh Hong Yu\anaconda3\envs\gpu_env\lib\site-packages\keras\initializers\initializers_v2.py:120: UserWarning: The initializer RandomNormal is unseeded and being called multiple times, which will return identical values each time (even if the initializer is unseeded). Please update your code to provide a seed to the initializer, or avoid using the same initializer instance more than once.
  warnings.warn(
```

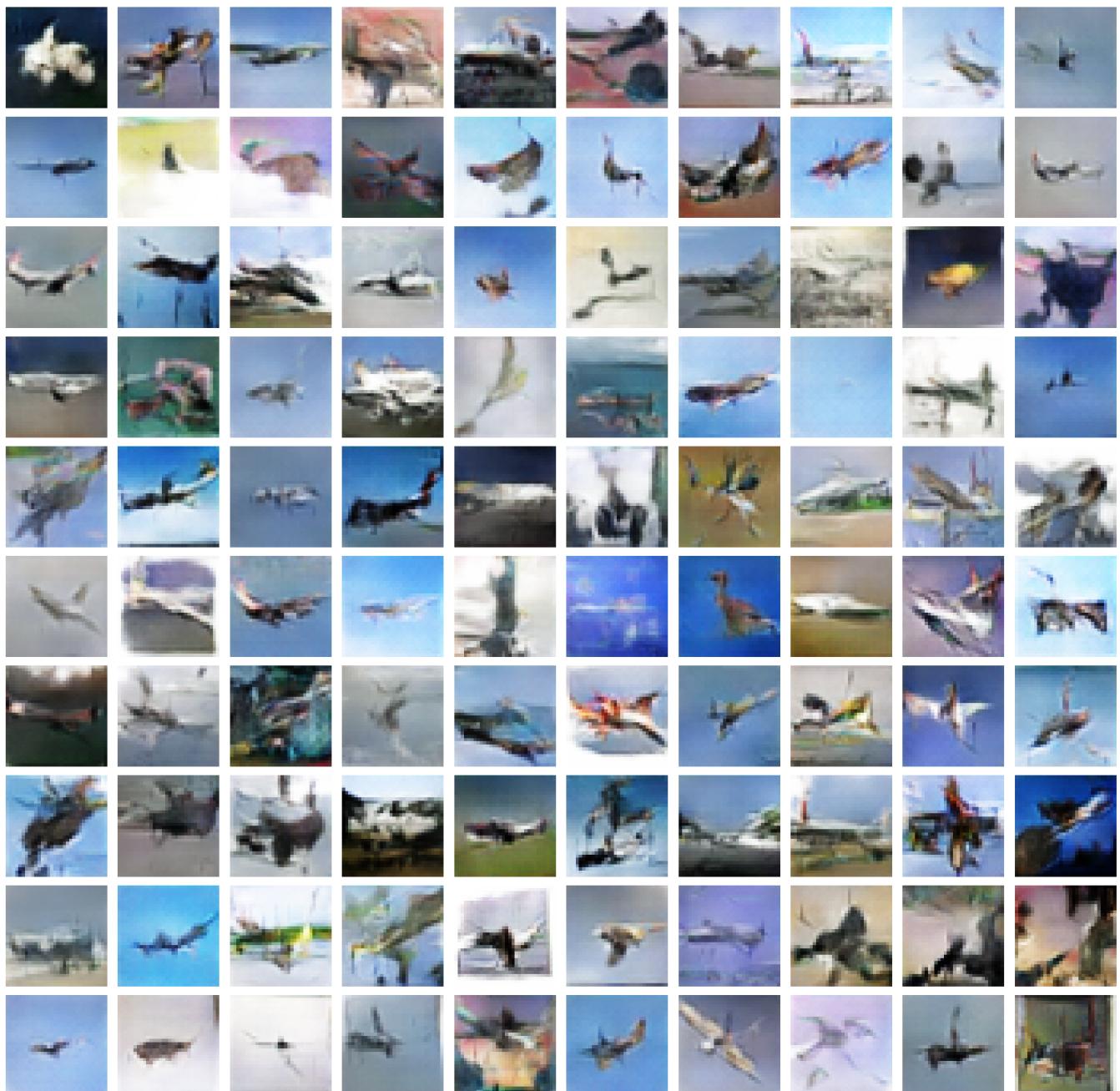
```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.  
Model: "generator_cGAN"
```

Layer (type)	Output Shape	Param #	Connected to
=====			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
=====			
=====			
Model: "generator_cGAN"			
=====			
Layer (type)	Output Shape	Param #	Connected to
=====			
=====			
Latent_Noise_Vector_z (InputLayer)	[None, 128]	0	[]
Conditions_y (InputLayer)	[(None, 10)]	0	[]
concatenate_10 (Concatenate)	(None, 138)	0	['Latent_Noise_Vector_z[0][0]', 'Conditions_y[0][0]']
dense_8 (Dense)	(None, 2048)	284672	['concatenate_10[0][0]']
batch_normalization_126 (Batch Normalization)	(None, 2048)	8192	['dense_8[0][0]']
p_re_lu_12 (PReLU)	(None, 2048)	2048	['batch_normalization_126[0][0]']
reshape_3 (Reshape)	(None, 2, 2, 512)	0	['p_re_lu_12[0][0]']
Base_Generator (Sequential)	(None, 16, 16, 64)	2784320	['reshape_3[0][0]']
Output_Layer (Conv2DTranspose)	(None, 32, 32, 3)	3075	['Base_Generator[0][0]']
=====			
=====			
Total params: 3,082,307			
Trainable params: 3,076,419			
Non-trainable params: 5,888			

```
In [ ]: for i in range(10):  
    # creating conditions and noise  
    conditions = np.full(100, i)  
    conditions = tf.keras.utils.to_categorical(conditions, 10)  
    latent_z = np.random.normal(size=(100, 128))  
  
    imgs = aux_cond_gan_generator.predict([latent_z, conditions])  
  
    fig = plt.figure(figsize=(20, 20), tight_layout=True)  
    for idx, img in enumerate(imgs):  
        ax = fig.add_subplot(10, 10, idx+1)  
        ax.imshow((img + 1) / 2)  
        ax.axis('off')  
    fig.suptitle(list(class_labels.values())[i], y=1, fontsize=28)  
    fig.savefig(fname=".//images/cGAN/{}.jpeg".format(list(class_labels.values())[i]))
```

```
4/4 [=====] - 0s 8ms/step  
4/4 [=====] - 0s 6ms/step  
4/4 [=====] - 0s 6ms/step  
4/4 [=====] - 0s 8ms/step  
4/4 [=====] - 0s 11ms/step  
4/4 [=====] - 0s 8ms/step  
4/4 [=====] - 0s 12ms/step  
4/4 [=====] - 0s 11ms/step  
4/4 [=====] - 0s 10ms/step  
4/4 [=====] - 0s 10ms/step
```

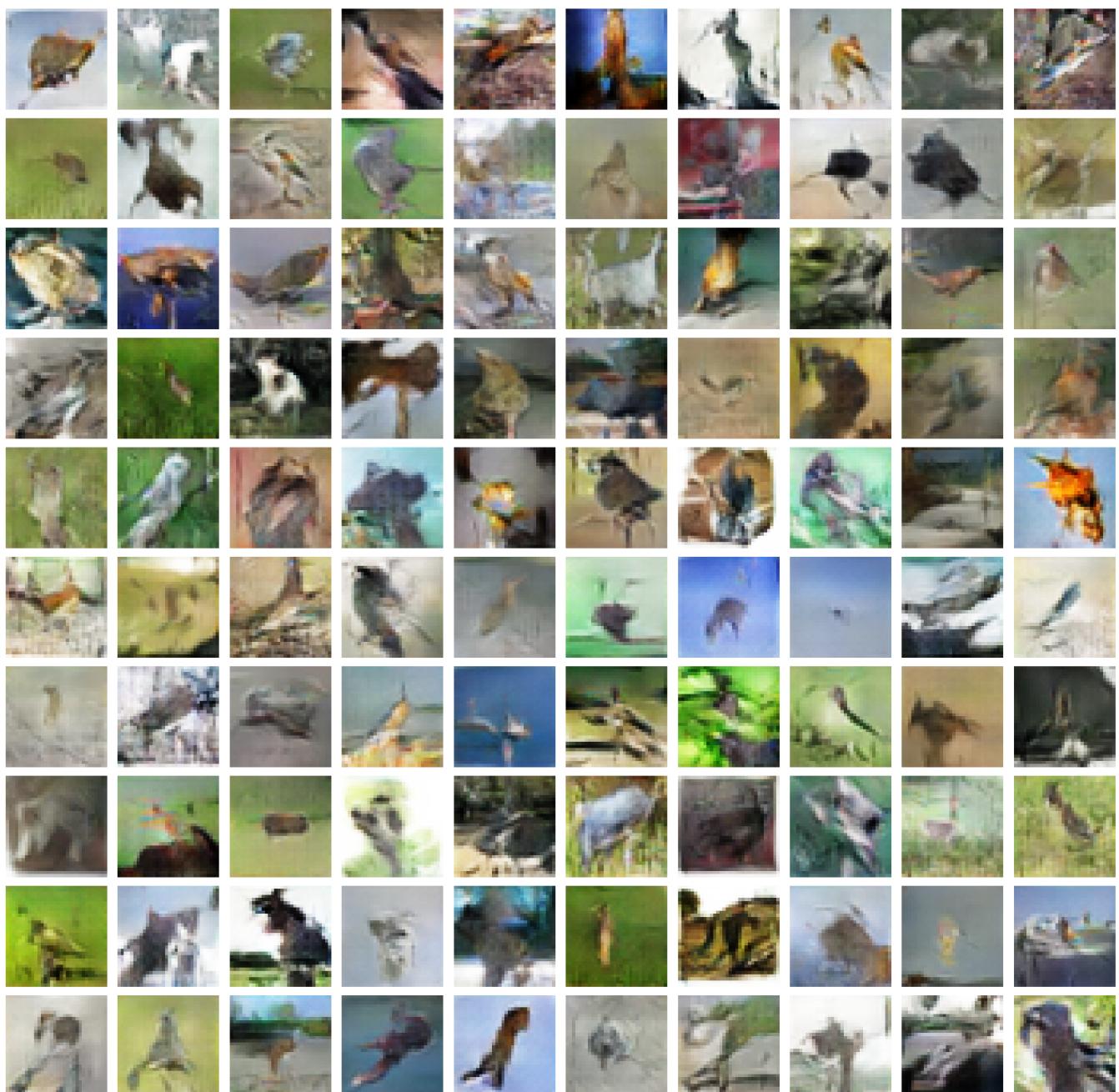
Airplane



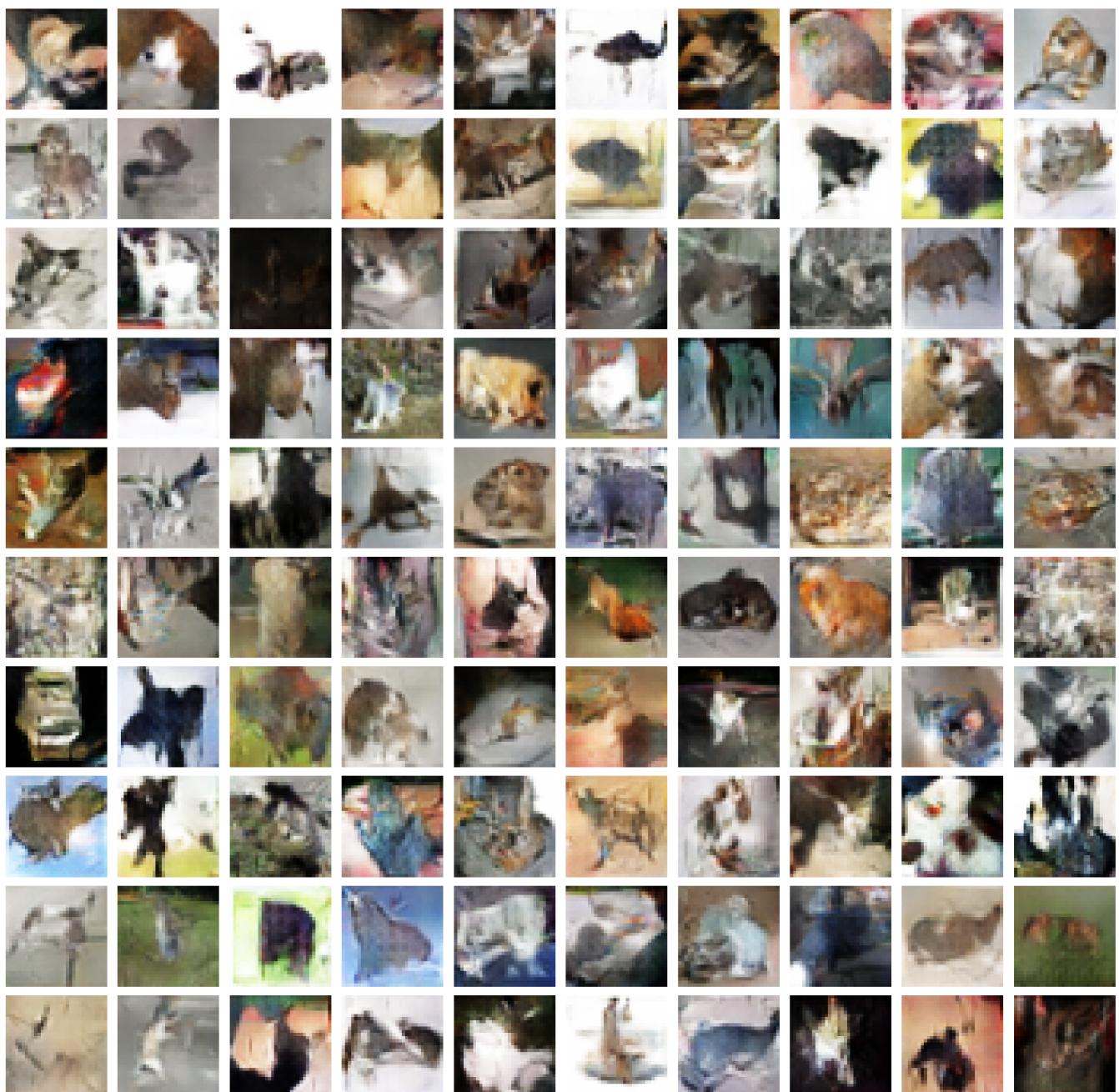
Automobile



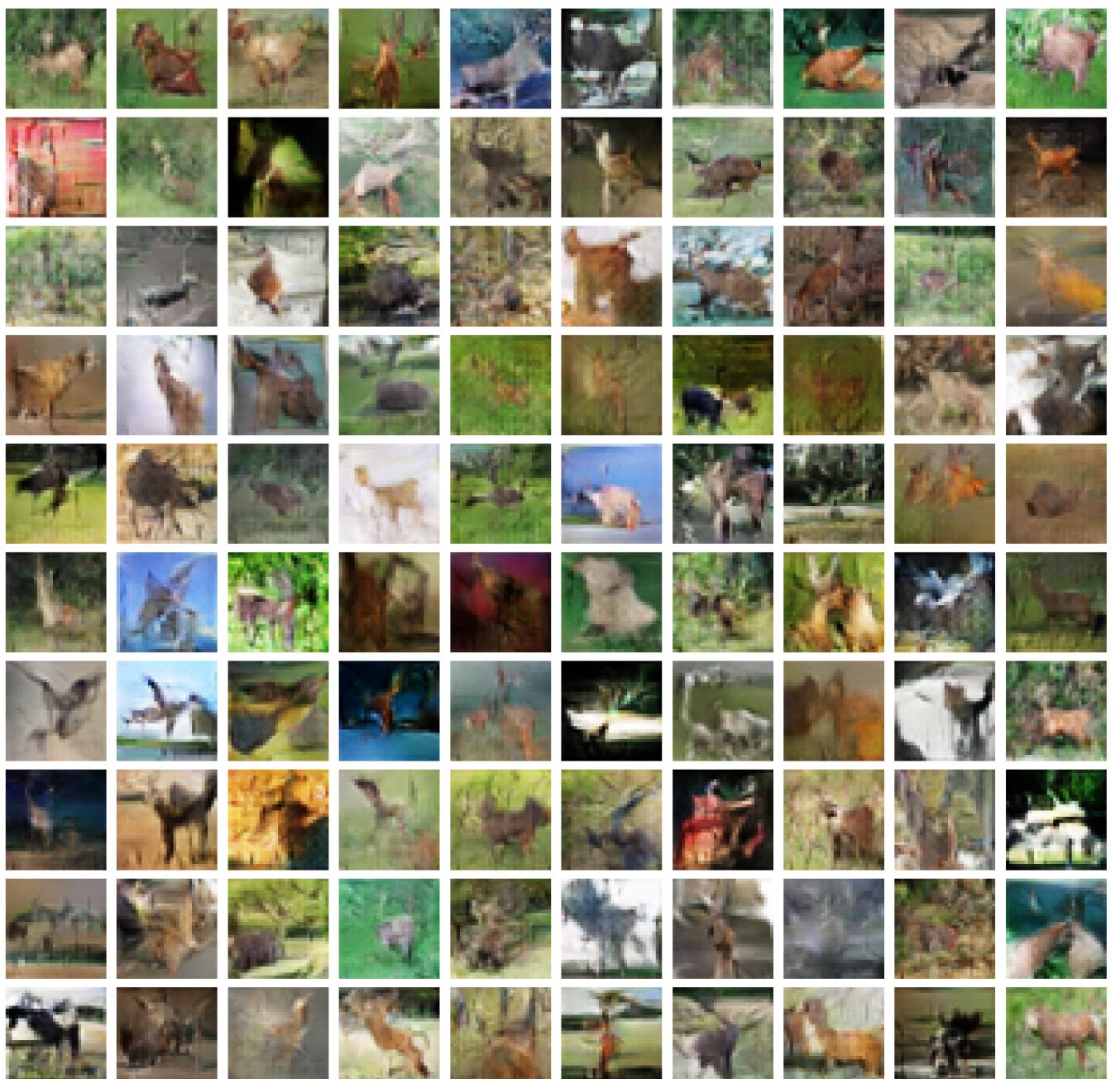
Bird



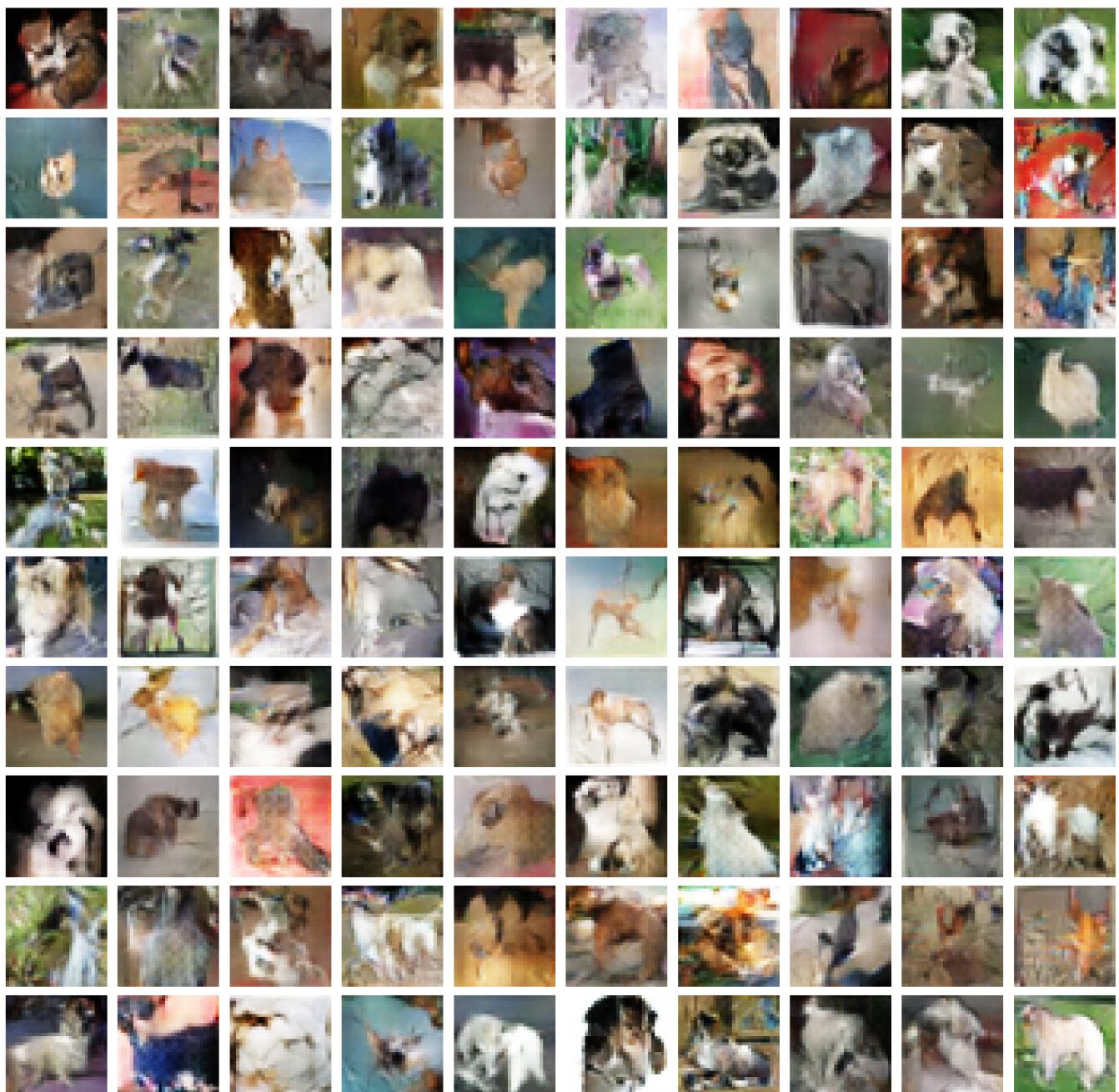
Cat



Deer



Dog



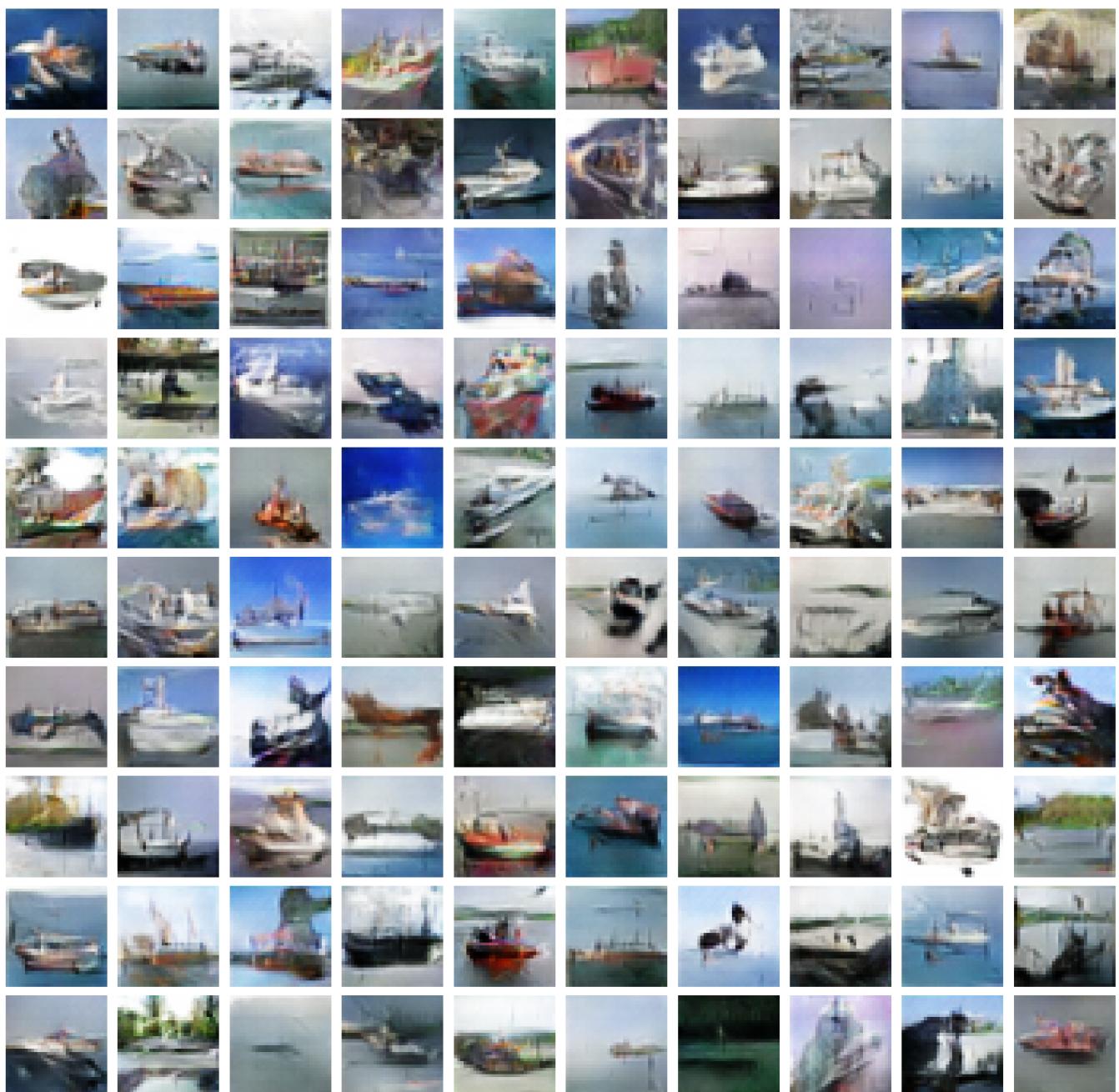
Frog



Horse



Ship



Truck

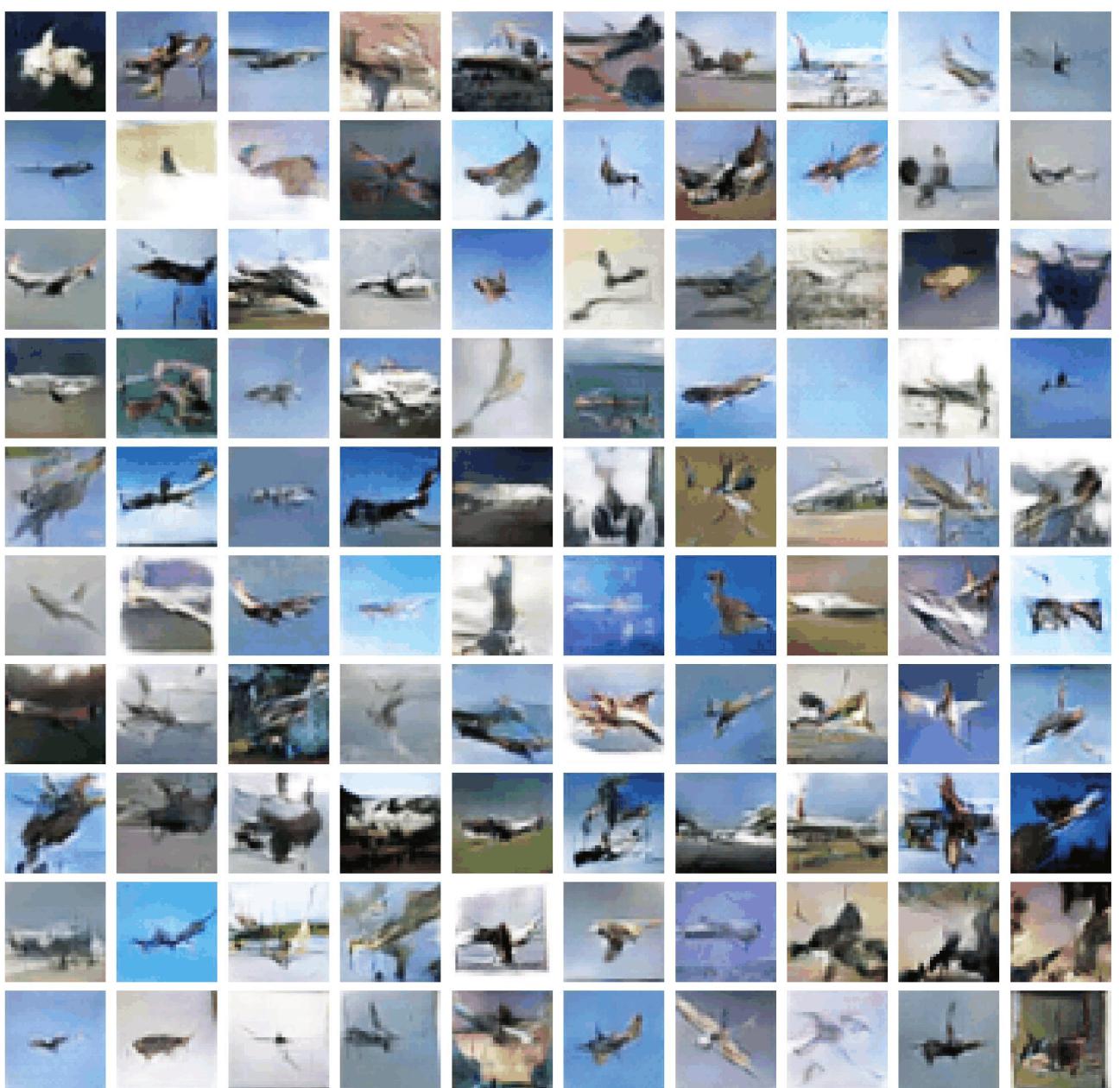


```
In [ ]: images = []
gan_img_paths = glob.glob("./images/cGAN/*.jpeg")
for path in gan_img_paths:
    images.append(imageio.imread(path))
imageio.mimsave('./images/cGAN/cGAN.gif', images, duration=1)
display_gif('./images/cGAN/cGAN.gif')
```

C:\Users\Soh Hong Yu\AppData\Local\Temp\ipykernel_14940\2200930972.py:4: DeprecationWarning:
Starting with ImageIO v3 the behavior of this function will switch to that of iio.v3.imread.
To keep the current behavior (and make this warning disappear) use `import imageio.v2 as imageio` or call `imageio.v2.imread` directly.

```
    images.append(imageio.imread(path))
```

Out[]:



Summary

We note that most of the images generated are fairly realistic, but some of the images have distorted features.

There are some issues with the images generated by the models.

1. Some of the wheels of the vehicles are warped or have no wheels present at all.
2. Some of the animals like horses and deer have thin or fat bodies.
3. We also note that some of the images are very corrupted by the background and noise.
4. Most of the animals have very weird parts combined together

We also want to note that the model has a much easier time with vehicles generation than the other classes. We also note that classes like cats, dogs and birds have learnt the texture of the animal but some of the shapes are warped. This is likely due to the variety of images and styles that the classes have and the GAN model learning and having a hard time to forms the parts of the animal.

Comparing final_model VS other people's model

Based on PapersWithCode.com, our final model is very far away from the FID score. We note that even though the ACGAN's FID score is 88.7613090961658 which is very far away from the State of the Art Image Generation on Cifar-10 [Lowest FID is 1.77]. [<https://paperswithcode.com/sota/image-generation-on-cifar-10>]