DELE CA2

Tan Yu Hoe	**P2026309**
Diploma in Applied AI and Analytics	DAAA/FT/2B/04
School of Computing	Singapore Polytechnic
tanyh.20@ichat.sp.edu.sg	
DELE	ST1504
CA2 Assignment	

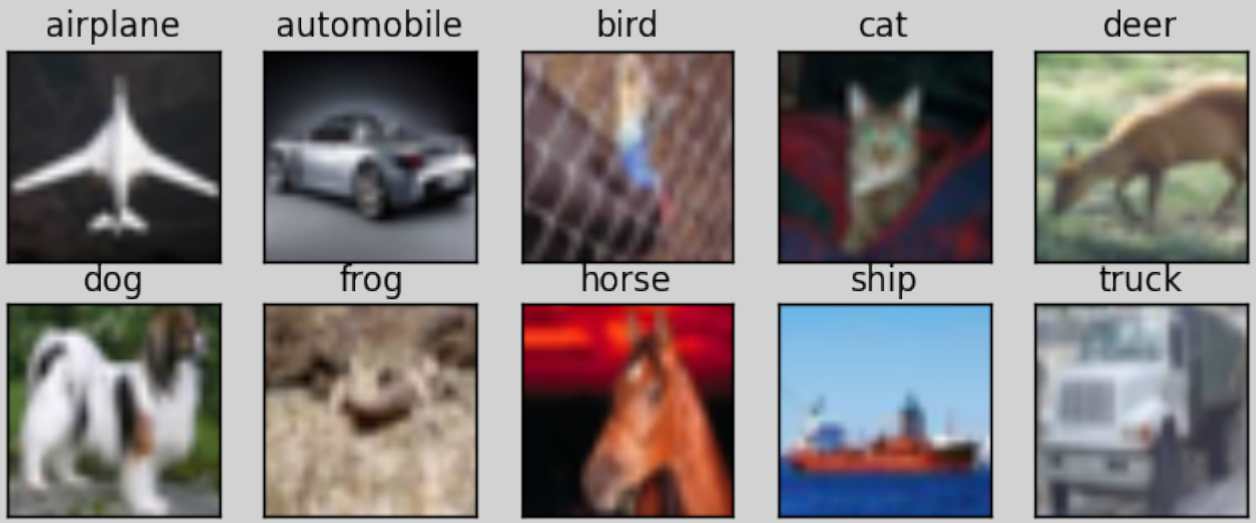
Table of Contents

Methodology and Motivation

In this assignment, Part A requires us to build and train a GAN architecture to create **1000 small colour images**, based on the CIFAR 10 dataset.

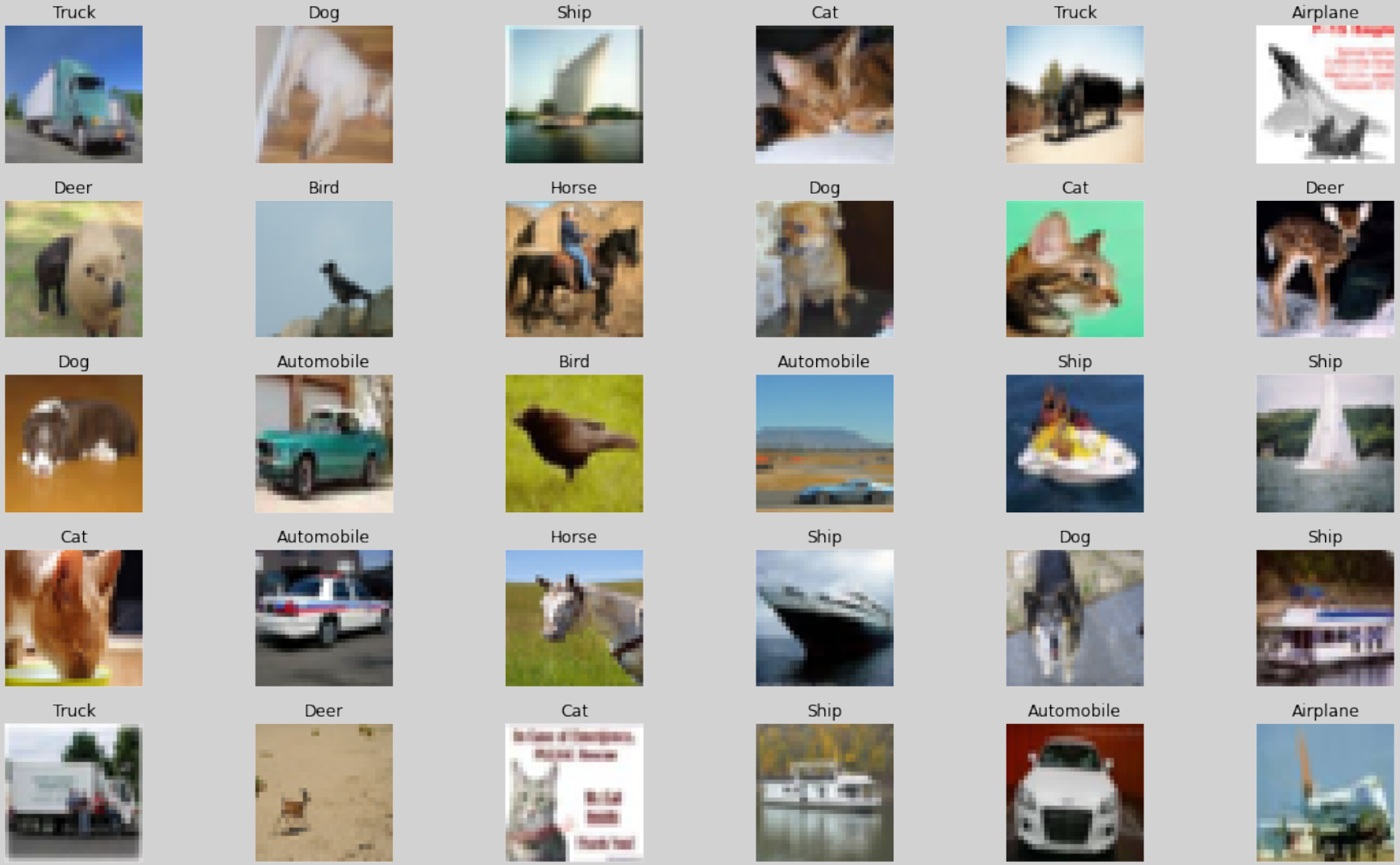
To attempt this task, I will be exploring various GAN architectures such as CGAN and ACGAN produce synthetic images of the CIFAR 10 dataset of substantiable quality.

CIFAR-10



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 and consists of 60,000 instances - 32 by 32 coloured images from 10 different classes. There are 50,000 training examples and 10,000 examples.

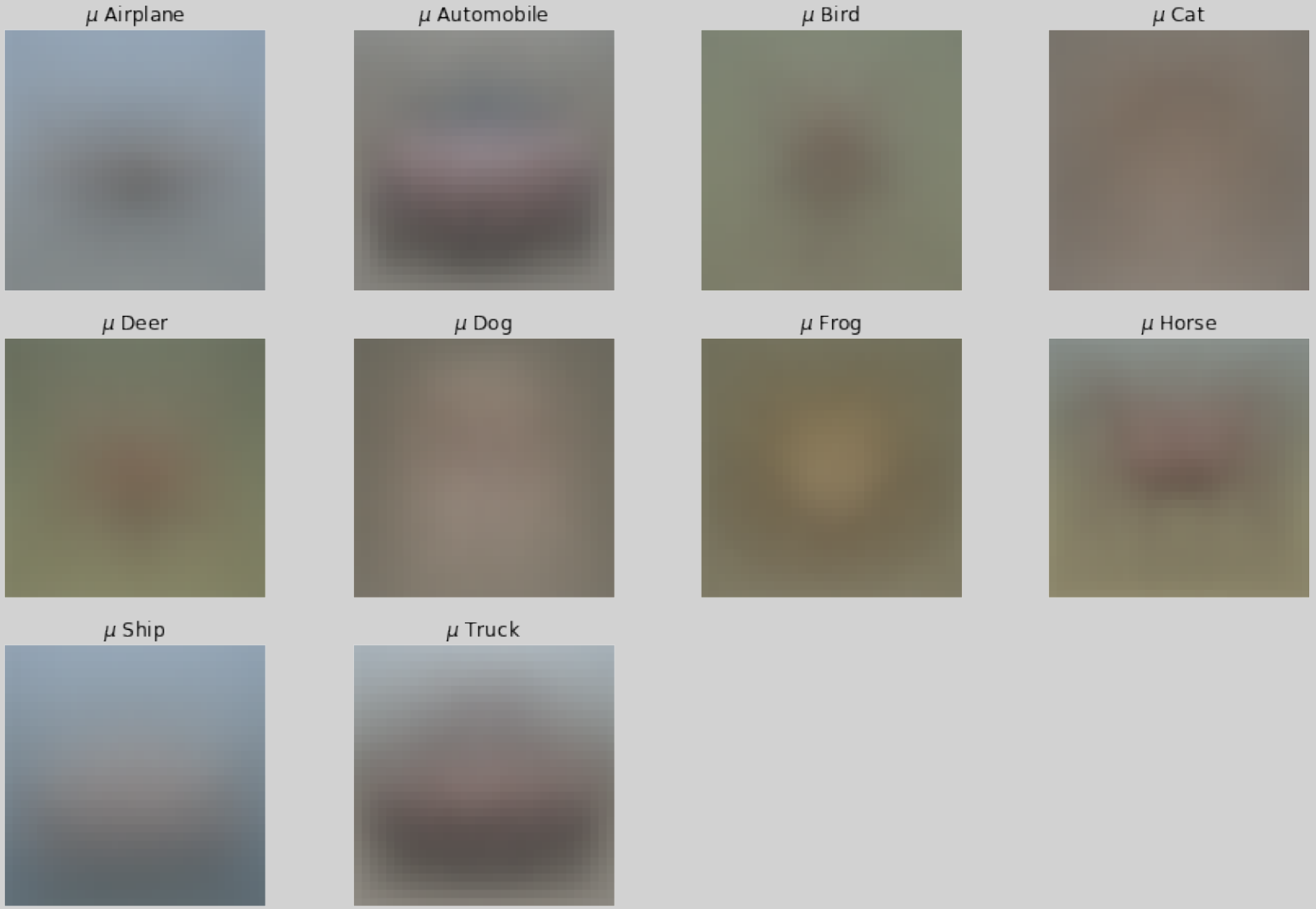
CIFAR-10


Truck
Dog
Ship
Cat
Truck
Airplane
Deer
Bird
Horse
Dog
Cat
Deer
Dog
Automobile
Bird
Automobile
Ship
Ship
Cat
Automobile
Horse
Ship
Dog
Ship
Truck
Deer
Cat
Ship
Automobile
Airplane

CIFAR-10


Training Class Distribution
count
5000
4000
3000
2000
1000
0
0
1
2
3
4
5
6
7
8
9

CIFAR-10


μ Airplane
μ Automobile
μ Bird
μ Cat
μ Deer
μ Dog
μ Frog
μ Horse
μ Ship
μ Truck

Generative Adversarial Networks on CIFAR-10

CIFAR-10 is a dataset of classes, with images and labels provided. I went to research for papers of the following properties:

- Generative Adversarial Networks utilising Labels
- Improvements on reliability on Generative Adversarial Networks training

Generative Adversarial Networks on CIFAR-10

For the first property, I found a 2014 paper on *"Conditional Adversarial Net"*. [1] The paper discussed about using label y as "conditions", the term coming from conditional probability $P(A|B)$, to be fed into the generator and discriminator.

The results from the paper have shown that using conditions for image generation have greatly aid in image generation for MNIST digits dataset, outperforming non-conditional adversarial networks during that era.

1. Mirza, M., & Osindero, S. (2014). Conditional Generative Adversarial Nets. arXiv [cs.LG]. Opgehaal van http://arxiv.org/abs/1411.1784

Generative Adversarial Networks on CIFAR-10

For the second property, I found a paper from 2016 *on "Conditional Image Synthesis with Auxiliary Classifier GANs"*. [1] The paper describes about extending a CGAN architecture to an AC-GAN architecture, by adding label conditioning.

The author explained that adding label conditioning improved image discrimination by ignoring the component of loss arising from class labels when there are no labels for training unsupervised GAN.

1. Odena, A., Olah, C., & Shlens, J. (2017). Conditional Image Synthesis With Auxiliary Classifier GANs. arXiv [stat.ML]. Opgehaal van http://arxiv.org/abs/1610.09585

Pixel Normalization / Rescaling

The purpose of normalisation in image processing is to “attempt” to bring the pixels down to a normal distribution $N(0,1)$, to mitigate the strong influence of very large or very small pixels. Here, I attempt to use **Min-Max Normalization** (also called Unity-Based Normalization), to bring the pixel values down to range $[0, -1]$. This is a necessary pre-processing step to improve better optimization within the neural network.

$$X' = a + \frac{(X - X_{min})(b - a)}{X_{max} - X_{min}} = -1 + \frac{X}{127.5}$$

where a and b are lower and upper bound of a predefined range.

For GAN, it is recommended to use hyperbolic tangent **tanh** activation function in the output layer from the generator model. Given the saturating activation function provides an output range of $[-1, 1]$, it would be recommended to scale down the image to a range $[-1, 1]$.

Deep Convolutional GAN

I started with a **Deep Convolutional GAN** (DCGAN) as a Baseline Model. The architecture consists of two neural networks – (1) a **generator** and (2) a **discriminator**.

1. **Generator** $\boldsymbol{G}$: to generate the synthetic images $G(z)$ from a latent noise vector (z) [as a Gaussian Noise $N(0,1)$]

2. **Discriminator** $\boldsymbol{D}$: to discriminate synthetic images $G(z)$ from the training images x

Deep Convolutional GAN Discriminator

The discriminator is used to “classify” (the better term would be “discriminate”) between a Real Image and a Synthetic Image. The network structure follows a typical CNN architecture. I have made some changes to the network; however some changes were found to be beneficial while some are found to be disadvantageous.

- 2 by 2 Strided Convolutions
- Leaky ReLU Activation Function - $\alpha = 0.2$
- Batch Normalization
- Spectral Normalization
- Gaussian Weights Initialization - $N(0, 0.2)$

```
Model: "discriminator_GAN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 spectral_normalization (Spe  (None, 16, 16, 64)       3200
 ctralNormalization)

 batch_normalization (BatchN  (None, 16, 16, 64)       256
 ormalization)

 leaky_re_lu (LeakyReLU)     (None, 16, 16, 64)        0

 spectral_normalization_1 (S  (None, 8, 8, 128)        131328
 pectralNormalization)

 batch_normalization_1 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 leaky_re_lu_1 (LeakyReLU)   (None, 8, 8, 128)         0

 spectral_normalization_2 (S  (None, 4, 4, 256)        524800
 pectralNormalization)

 batch_normalization_2 (Batc  (None, 4, 4, 256)        1024
 hNormalization)

 leaky_re_lu_2 (LeakyReLU)   (None, 4, 4, 256)         0

 spectral_normalization_3 (S  (None, 2, 2, 512)        2098176
 pectralNormalization)

 batch_normalization_3 (Batc  (None, 2, 2, 512)        2048
 hNormalization)

 leaky_re_lu_3 (LeakyReLU)   (None, 2, 2, 512)         0

 global_max_pooling2d (Globa  (None, 512)              0
 lMaxPooling2D)

 dense (Dense)               (None, 1)                 513

=================================================================
Total params: 2,761,857
Trainable params: 2,758,977
Non-trainable params: 2,880
_________________________________________________________________
```

```python
# function to create discriminator model
def create_discriminator(image_size):
    weights_init = RandomNormal(mean=0, stddev=0.02)
    discriminator = Sequential([
        Input(shape=input_dim),

        ## 2 by 2 Strides for downsampling
        ## Leaky ReLU for Activation
        SpectralNormalization(
            Conv2D(64, kernel_size=4, strides=2, padding='same', kernel_initializer=weights_init)
        ),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),

        # Conv 2
        SpectralNormalization(
            Conv2D(128, kernel_size=4, strides=2, padding='same', kernel_initializer=weights_init)
        ),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),

        # Conv 3
        SpectralNormalization(
            Conv2D(256, kernel_size=4, strides=2, padding='same', kernel_initializer=weights_init)
        ),
        BatchNormalization(momentum=0.8),
        LeakyReLU(0.2),

        # # Conv 4
        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=input_dim).summary()
```

Deep Convolutional GAN Generator

The Generator can be thought of as a mapping function from a latent noise vector to a generated image/3D tensor. Some key features I have implemented:

- Transposed Convolution Layer
- Batch Normalization
- Gaussian weights Initialization
- PReLU Activation Function

```
Model: "generator_GAN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 2048)              264192

 reshape (Reshape)           (None, 2, 2, 512)         0

 conv2d_transpose (Conv2DTra  (None, 4, 4, 256)        2097408
 nspose)

 batch_normalization_4 (Batc  (None, 4, 4, 256)        1024
 hNormalization)

 p_re_lu (PReLU)             (None, 4, 4, 256)         4096

 conv2d_transpose_1 (Conv2DT  (None, 8, 8, 128)        524416
 ranspose)

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 p_re_lu_1 (PReLU)           (None, 8, 8, 128)         8192

 conv2d_transpose_2 (Conv2DT  (None, 16, 16, 64)       131136
 ranspose)

 batch_normalization_6 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 p_re_lu_2 (PReLU)           (None, 16, 16, 64)        16384

 conv2d_transpose_3 (Conv2DT  (None, 32, 32, 3)        3075
 ranspose)

=================================================================
Total params: 3,050,691
Trainable params: 3,049,795
Non-trainable params: 896
_________________________________________________________________
```

```python
# function to create generator model
def create_generator(latent_dim):

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

    generator = Sequential([
        # fc block: handling latent vector z
        Input(shape=(latent_dim,)),
        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(latent_dim=latent_dim).summary()
```

DCGAN Training Function

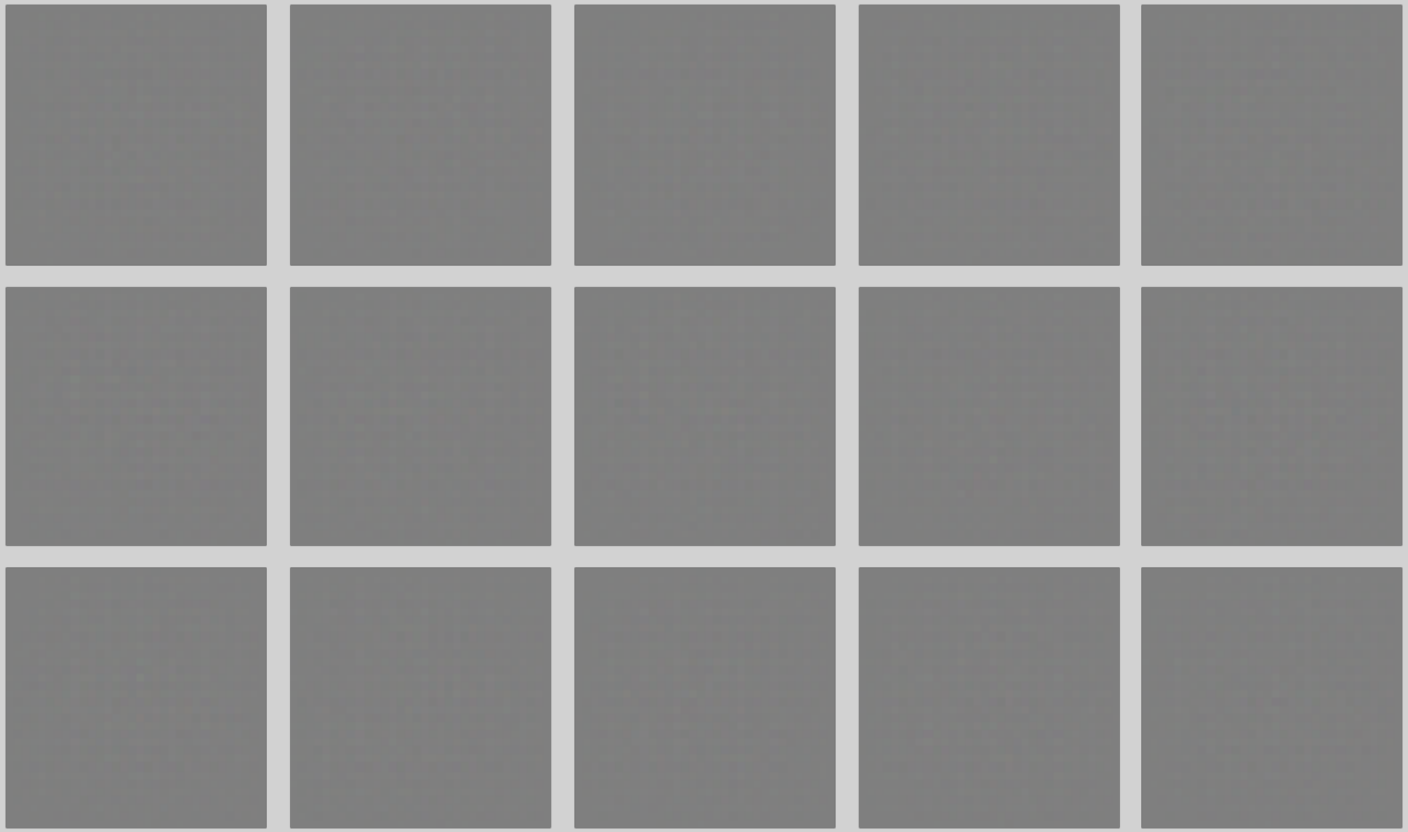
```
# Train the discriminator
with tf.GradientTape() as tape:
    predictions = self.discriminator(combined_images)
    d_loss = self.loss_fn(labels, predictions)
grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
self.d_optimizer.apply_gradients(
    zip(grads, self.discriminator.trainable_weights)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

```
# 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))
```

Deep Convolutional GAN Hyperparameters

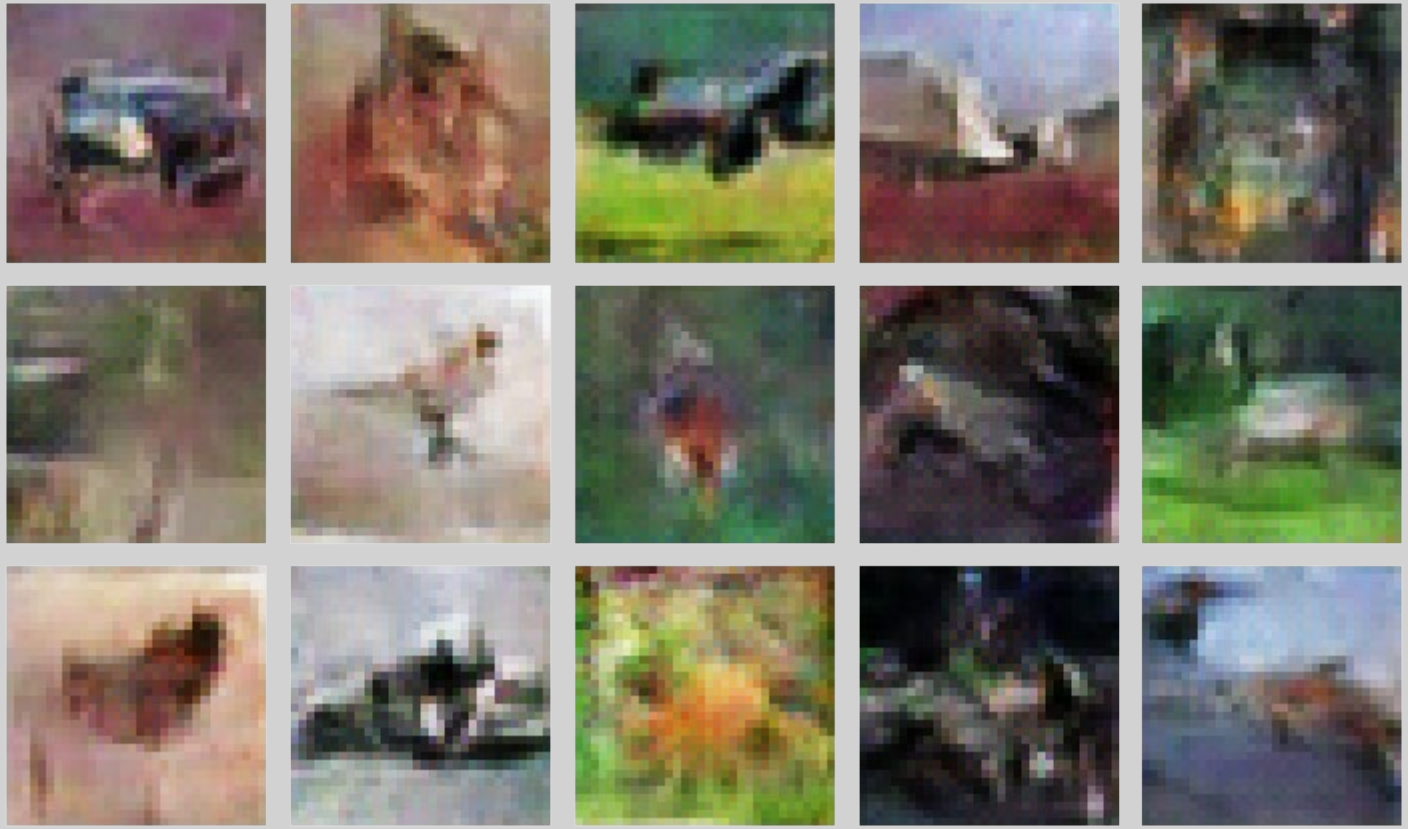
- Generator Optimizer : $Adam(\alpha = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999)$
- Discriminator Optimizer : $Adam(\alpha = 0.0002, \beta_1 = 0.5, \beta_2 = 0.999)$
- Batch Size : 64
- Latent Vector Dimensions : 128
- Epochs: 100

Deep Convolutional GAN
Epoch 0



Deep Convolutional GAN
Epoch 25

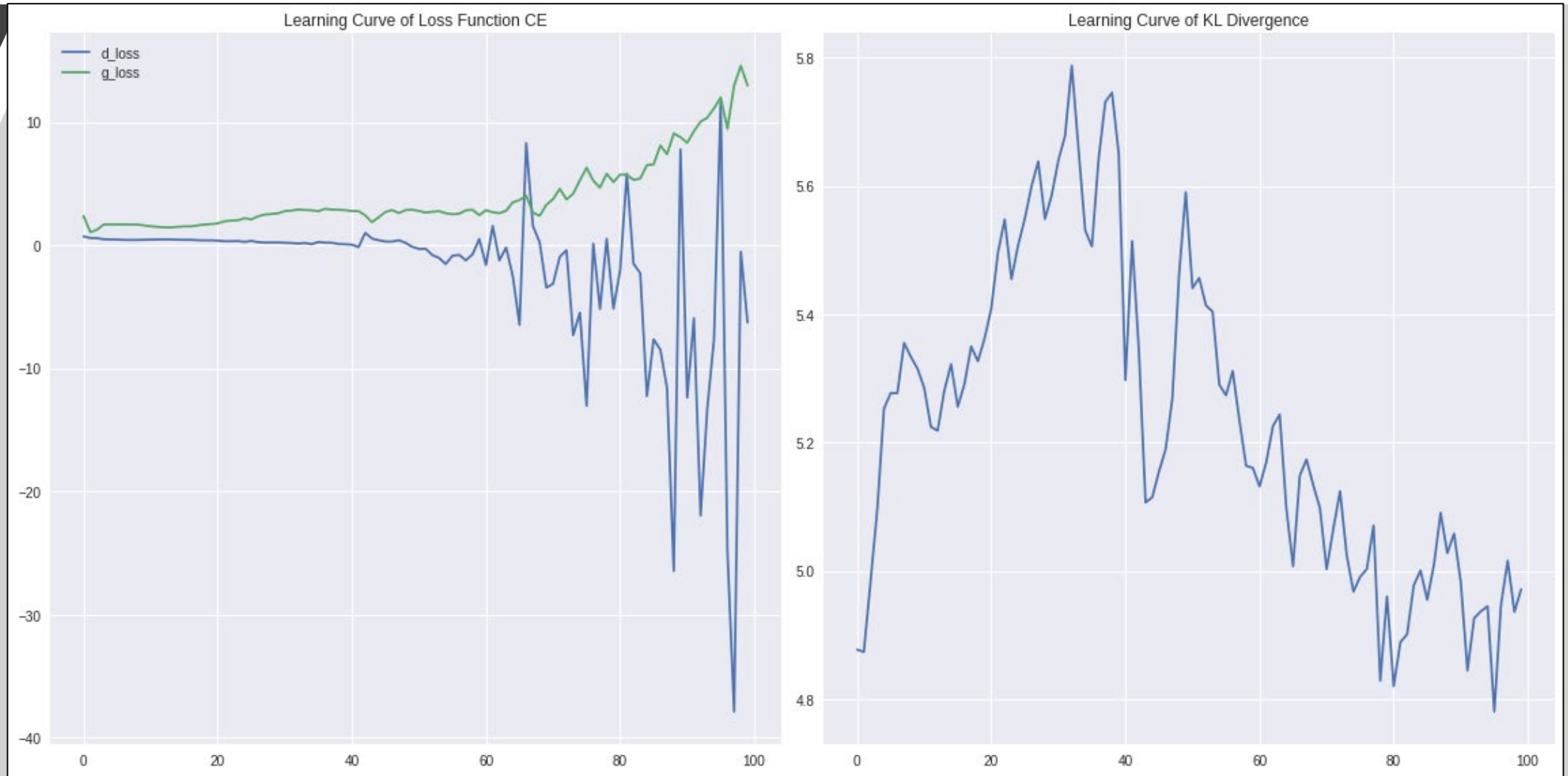
Deep Convolutional GAN
Epoch 50



Deep Convolutional GAN
Epoch 75

Deep Convolutional GAN
Epoch 100

Deep Convolutional GAN Learning Curve



Deep Convolutional GAN FID Evaluation

Fréchet Inception Distance is a metric used to access the quality of images created by the generator by comparing the distribution of the generated images with the distribution of real images that were used to train the generator.

To implement FID, the paper on *"GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium"* mentioned that the minimum number of generated sample is 10,000 in order to evaluate FID. I randomly sampled 10,000 images from the training set and generate 10,000 images and evaluate against each other.

FID Score (Epoch 40): 138.33 ± 0.837

Deep Convolutional GAN Quantitative Results

Architecture	Epoch Checkpoint	FID	KL Divergence	ResNet34 Accuracy
DCGAN	40	138.33 +- 0.837	5.6873	-

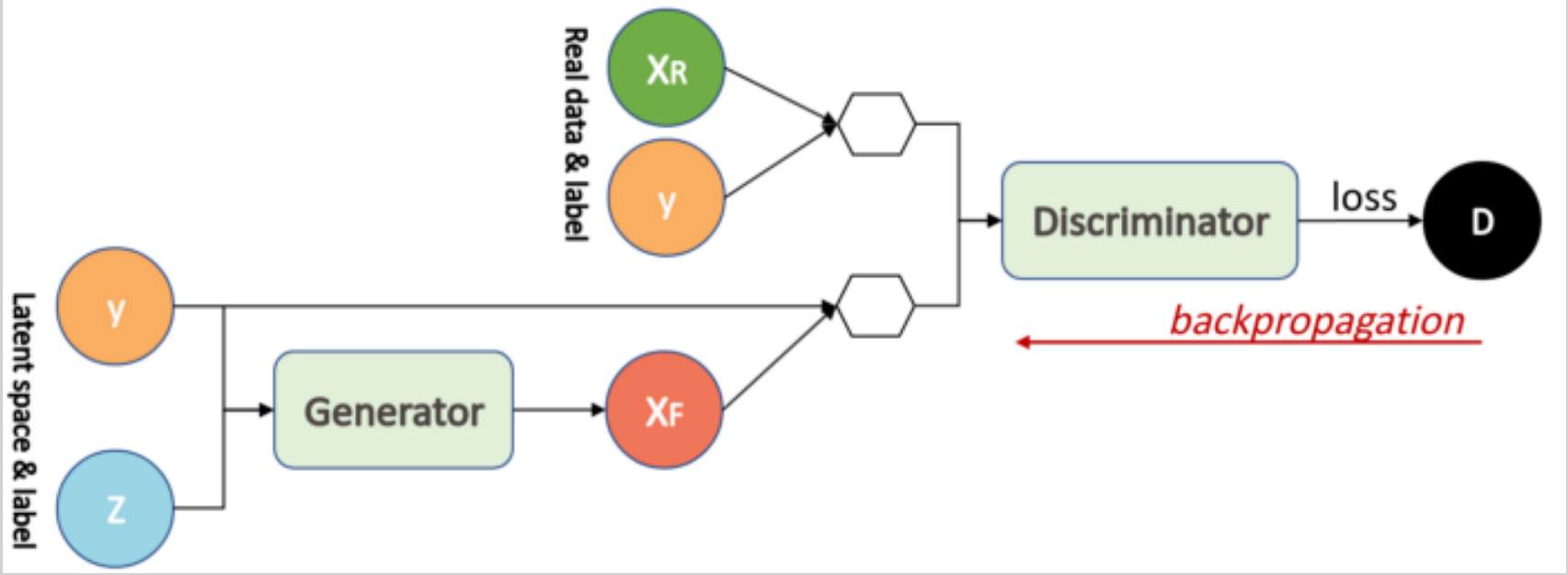
Deep Convolutional GAN
Conclusion and Further Improvements

- FID Score (Epoch 40): 138.33 ± 0.837
- FID is quite far from the benchmarks on Papers with Code
- Stable for 40 epochs before encountering a collapse.
- Generated images are not in clear quality, it is hard to identify the focal point of the image.
- Some images were found to be mixed with each other, like a cat and dog was transfigured together into a single image.
- Spectral Normalization and Weights Initialization could have too much regularization effect on the networks.

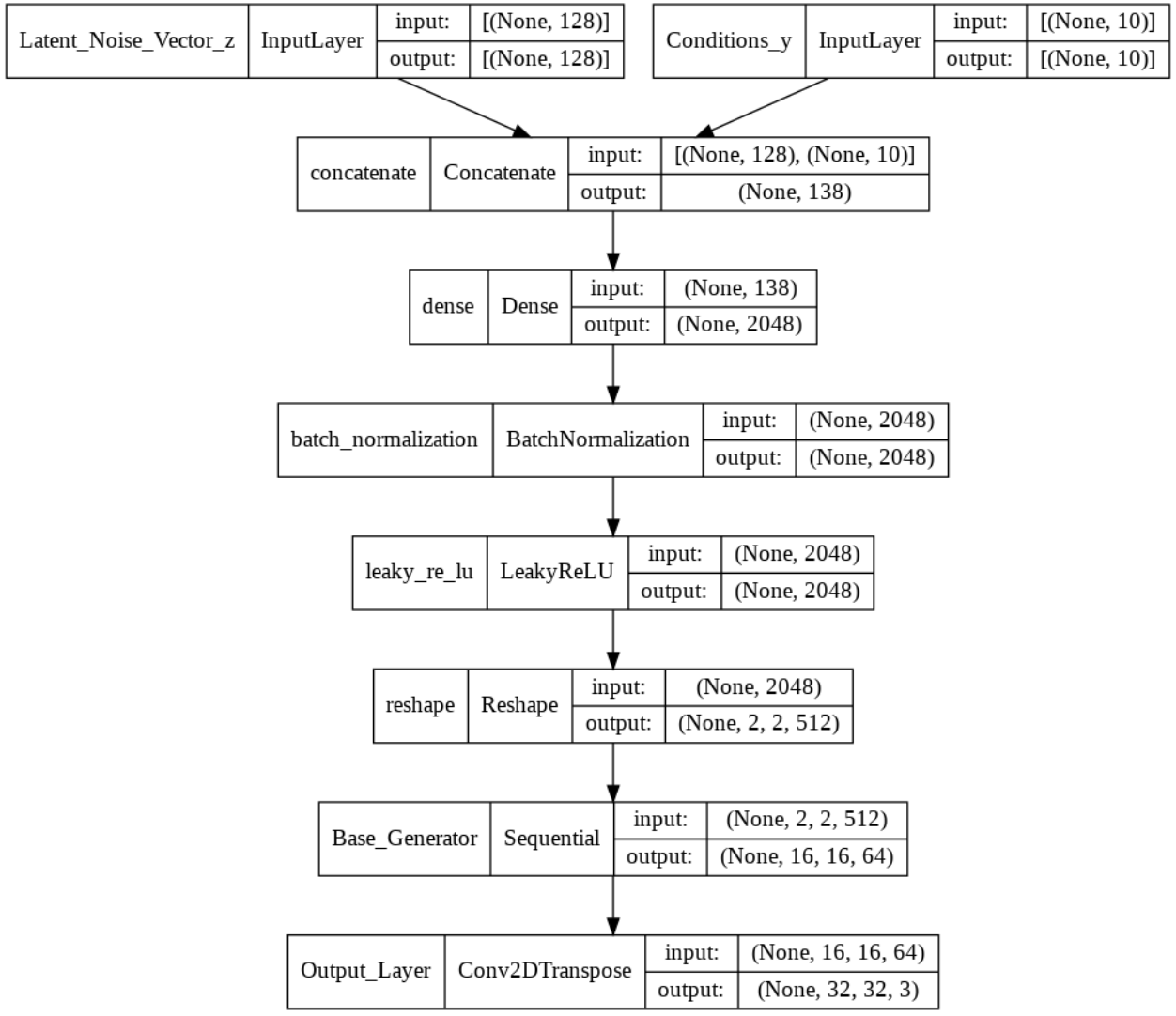
- Adding Labels into training could improve training and identification. (Conditional GAN, InfoGAN, ACGAN)
- Changing Loss Function could perhaps improve results. (WGAN, WGAN-GP, LSGAN)

Conditional GAN

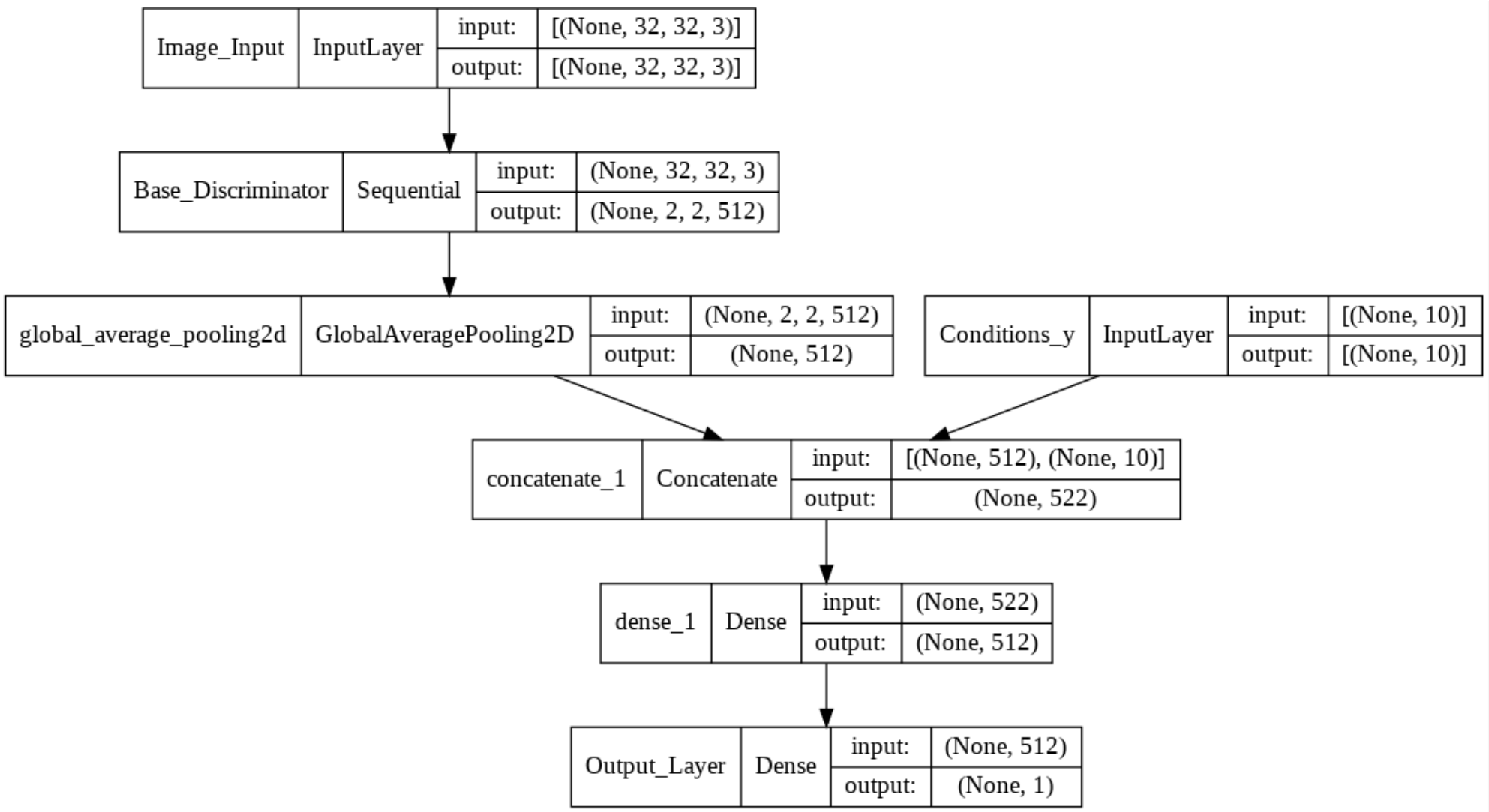
In a Generative Adverserial Network (unconditional), there is no control of the type of images generated. Simply put, it is providing an input to the Generative model and receiving a batch of randomly generated images. These types of GAN are not ideal for the CIFAR-10 dataset, as there are labels provided and we might as well make the full use of it.



Conditional GAN Generator

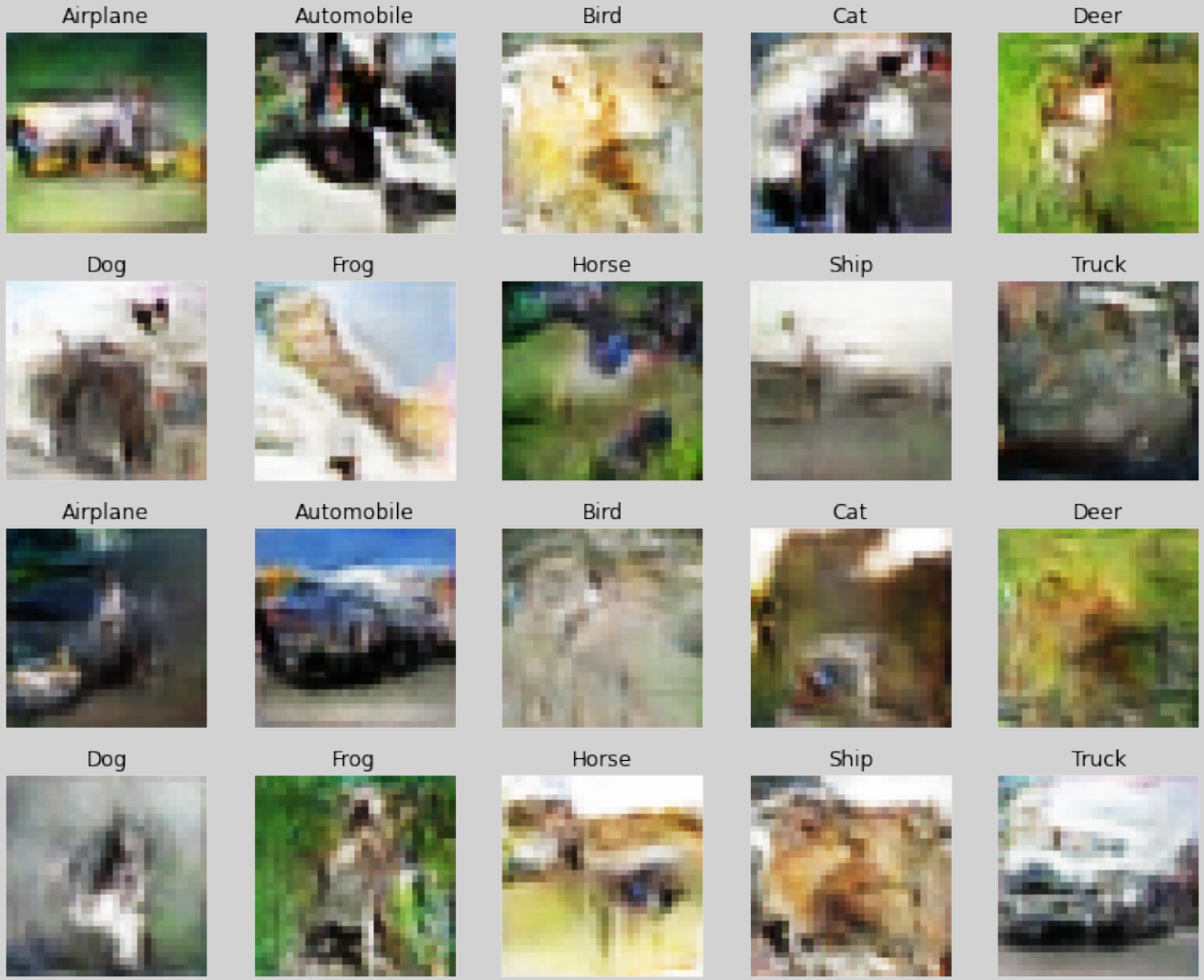


Conditional GAN Discriminator

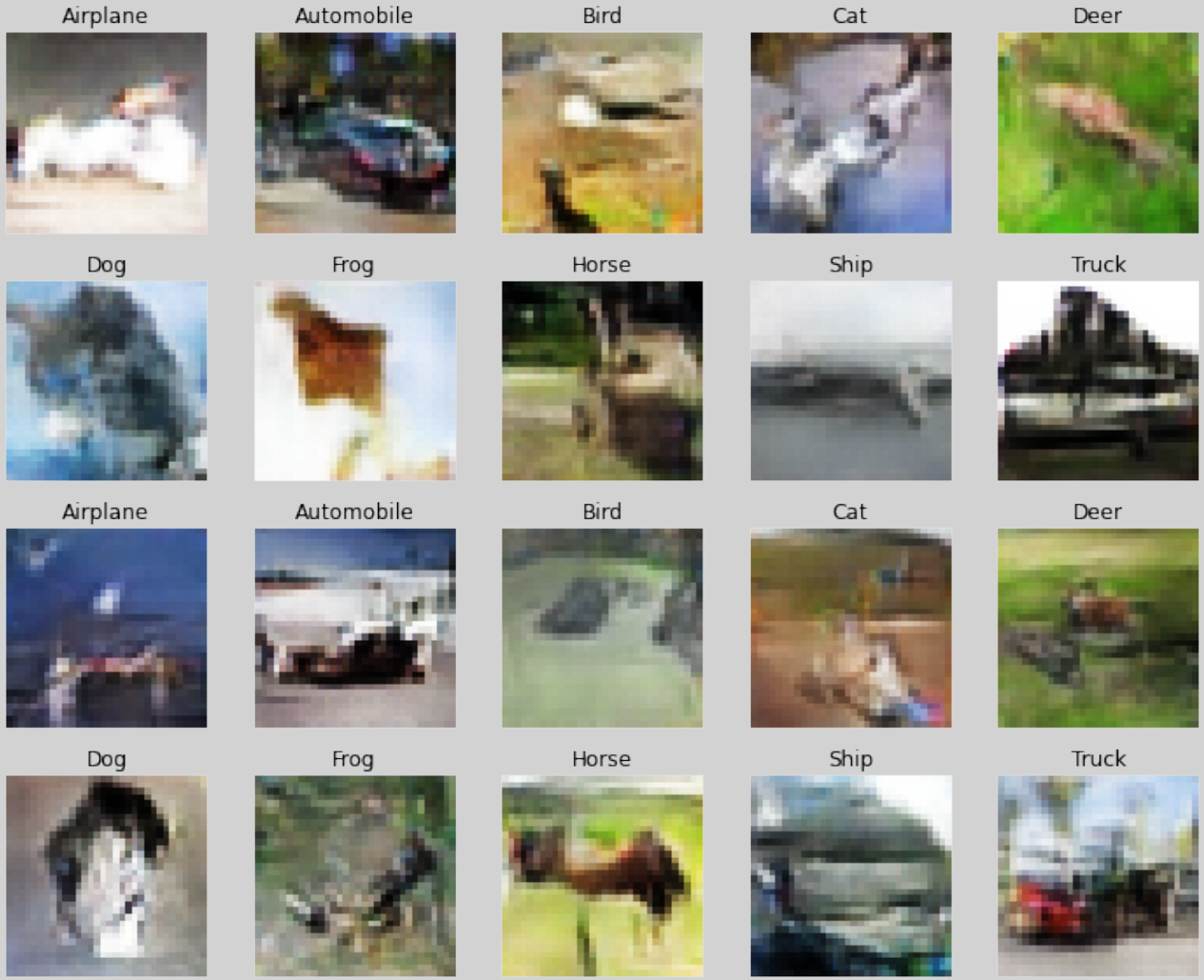


Conditional GAN
Epoch 0

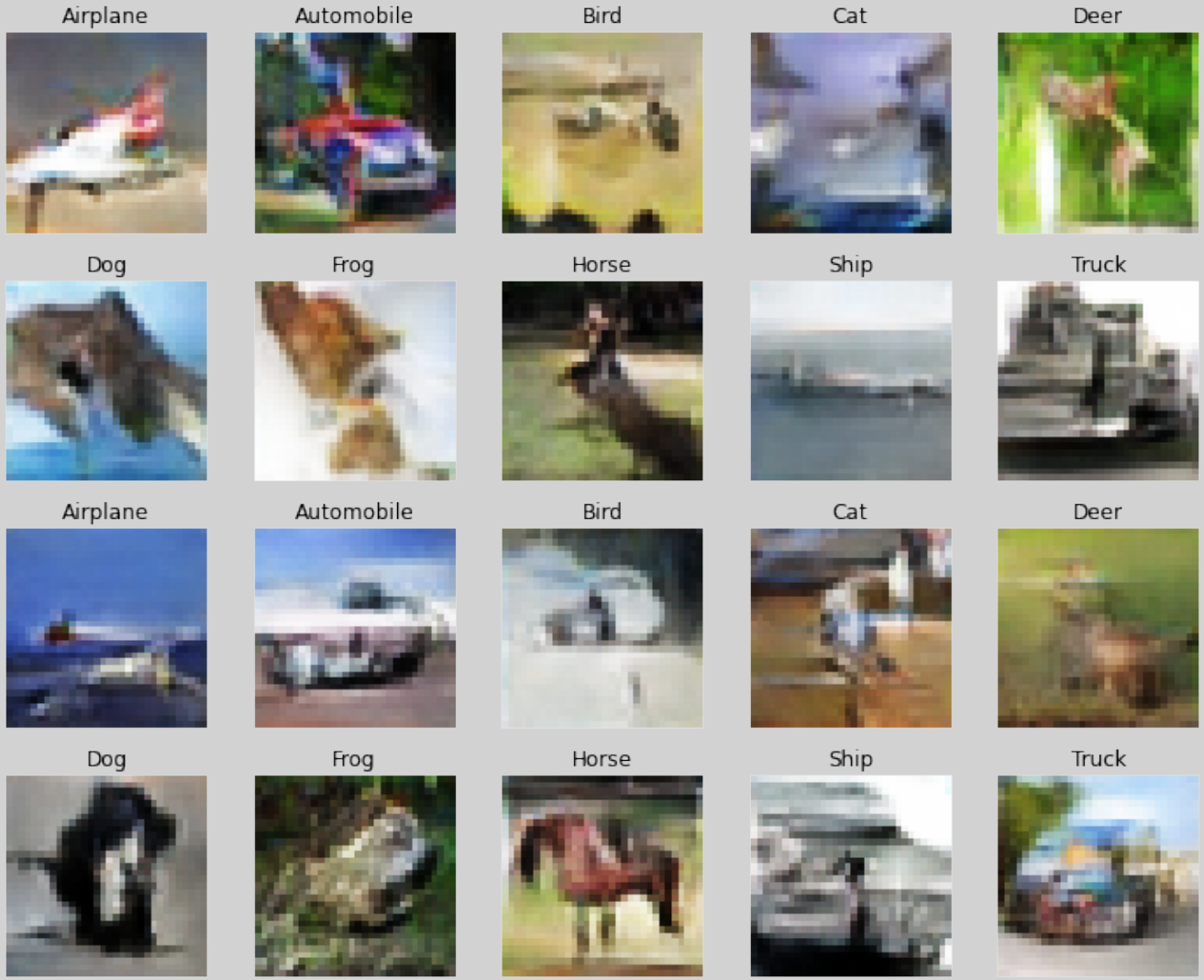
Conditional GAN
Epoch 20



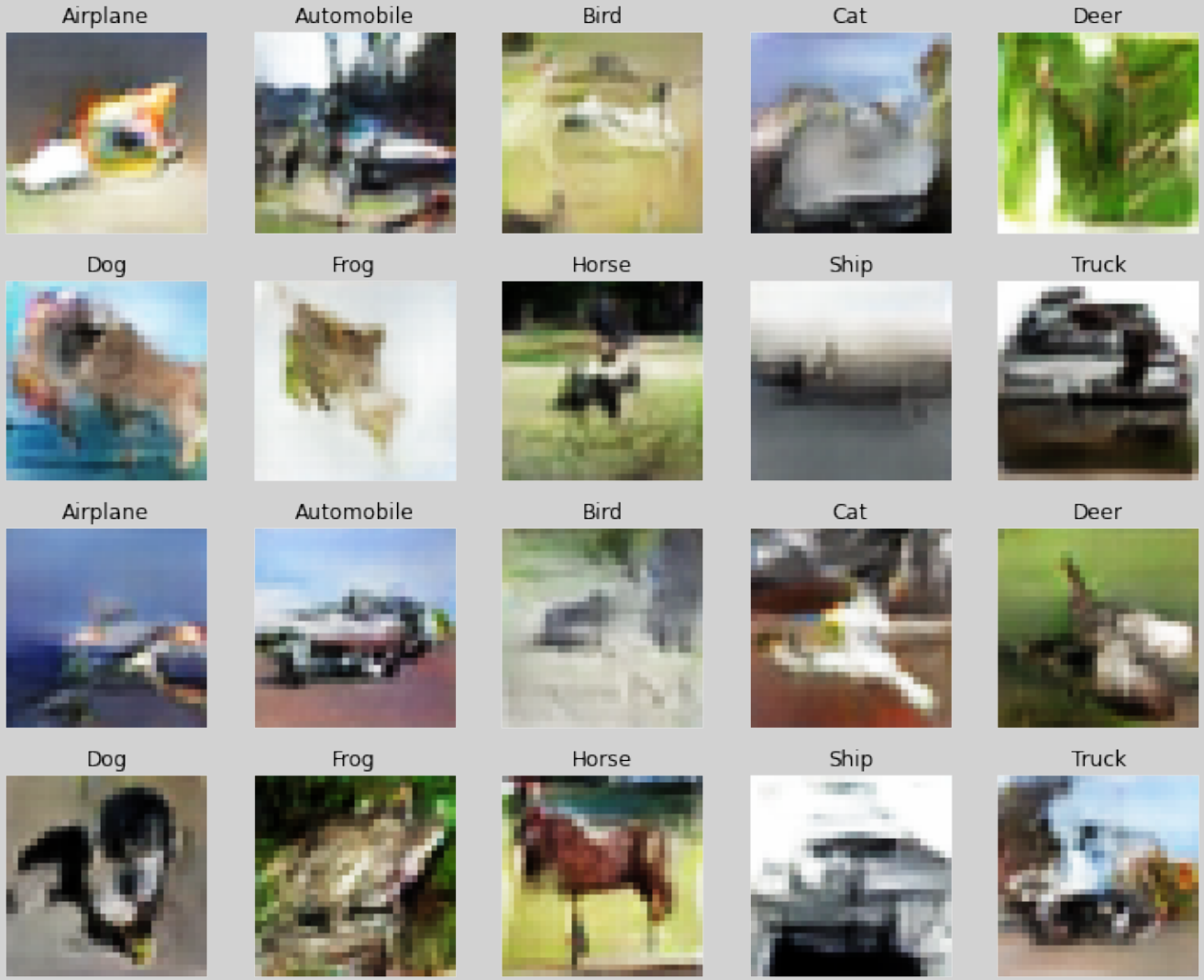
Conditional GAN
Epoch 40



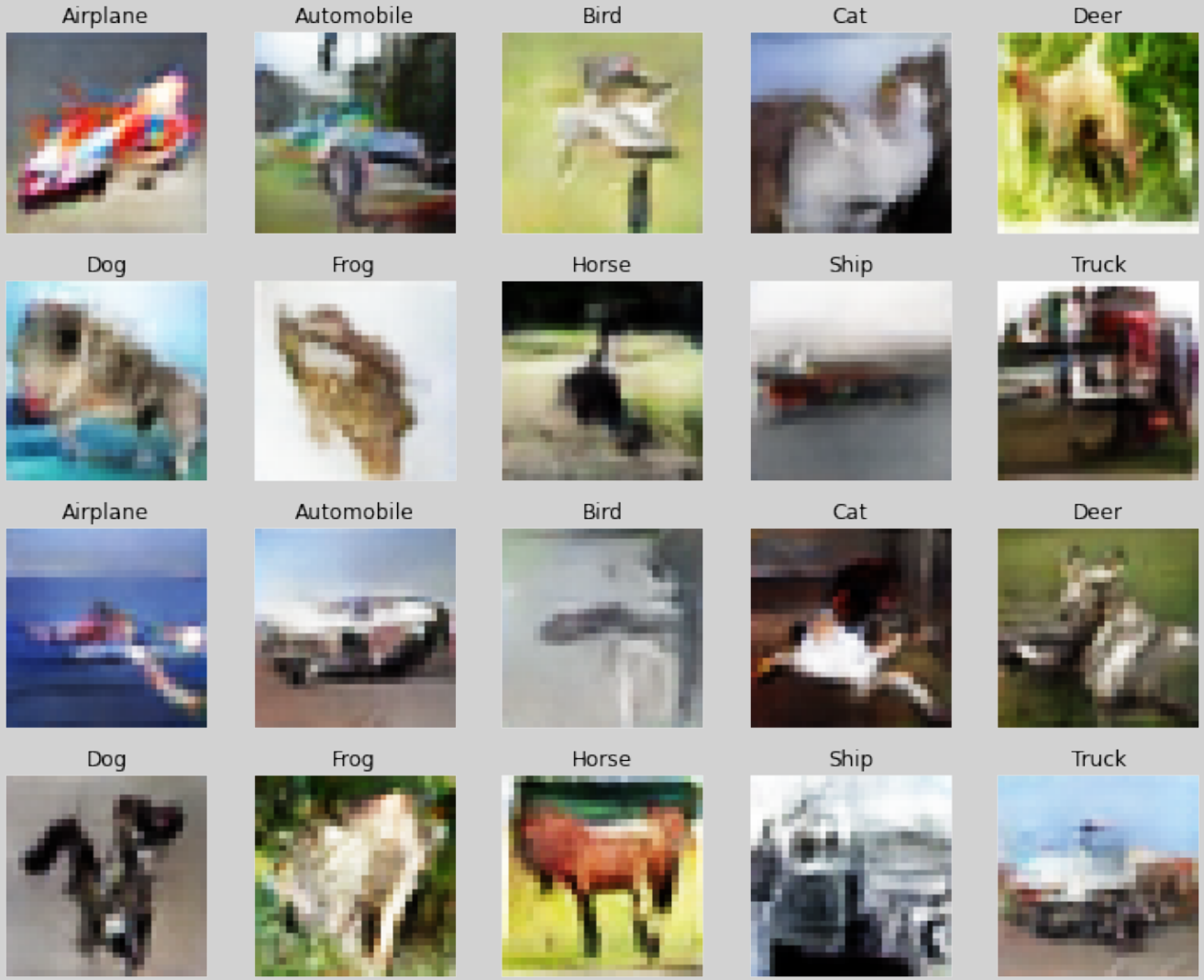
Conditional GAN
Epoch 60



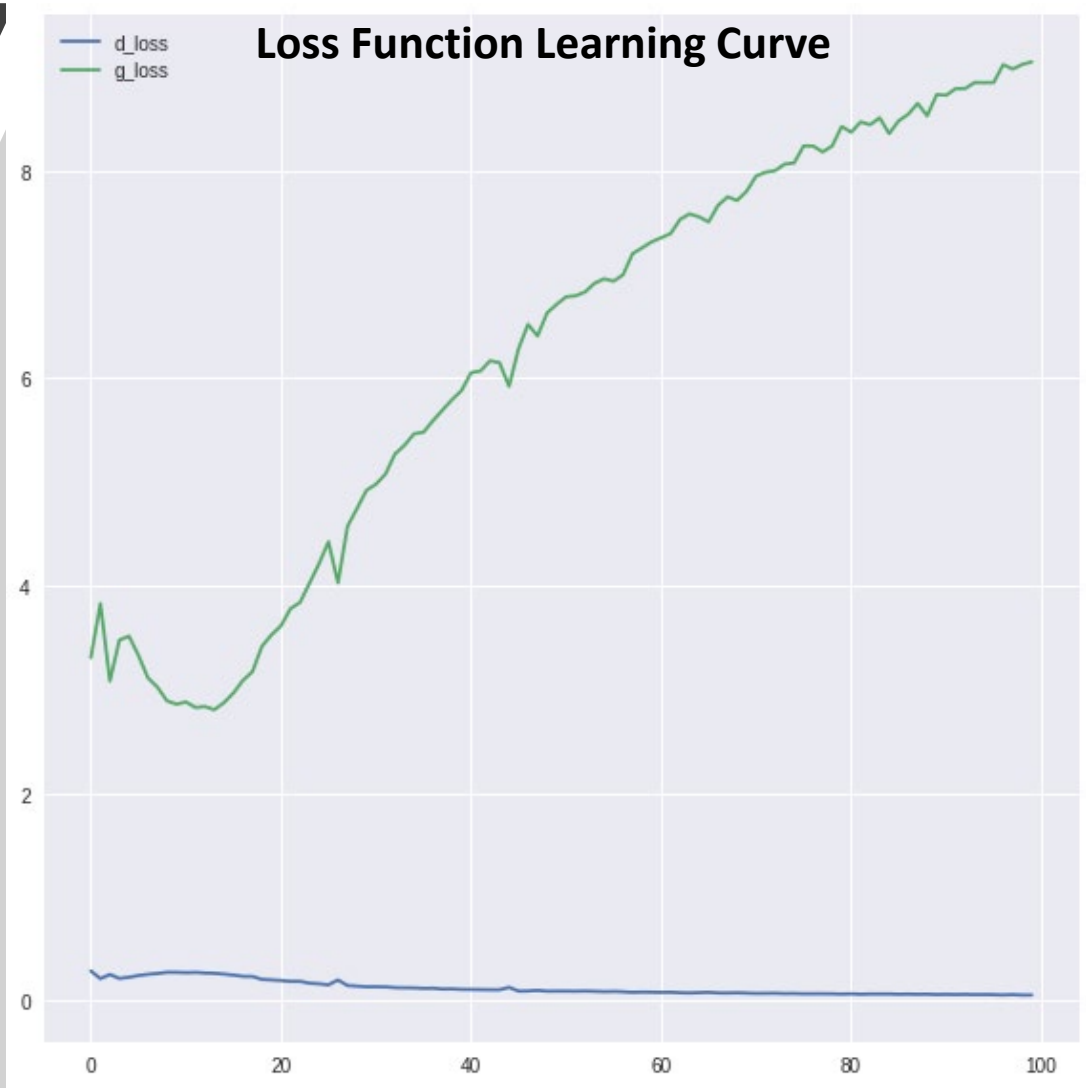
Conditional GAN
Epoch 80

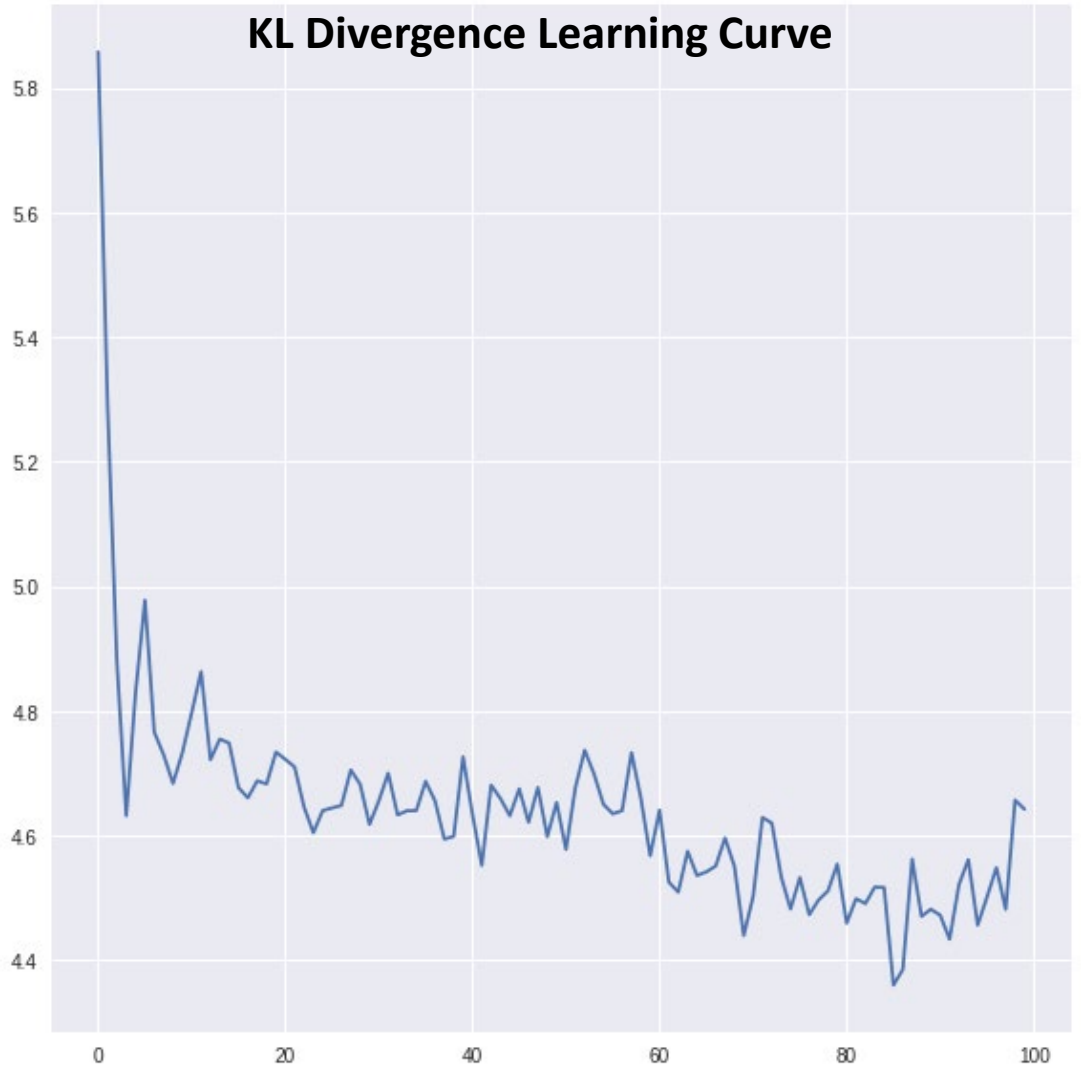


Conditional GAN
Epoch 100



Conditional GAN Learning Curve





Conditional GAN ResNet32 Evaluation

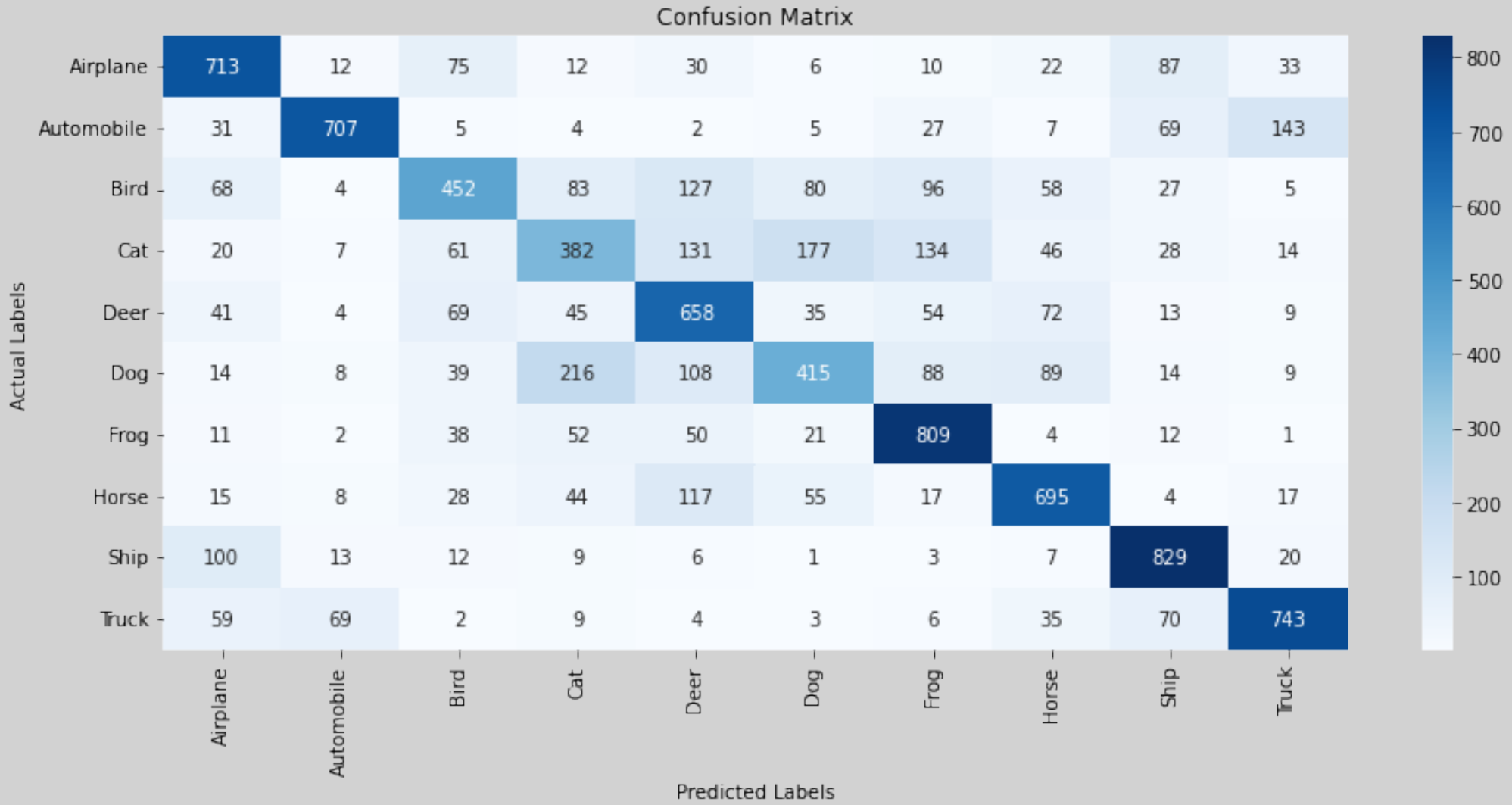
In my opinion, using FID may not be entire accurate in terms of evaluating our synthetic images, given that the pretrained weights from Inception were trained on the ImageNet dataset. Another issue was that I couldn't not tell which class is generated better than the other classes.

In order to resolves these issues, I've pointed out, I will be using an image classifier I've built during ST1501 based on the CIFAR-10 dataset as a surrogate model. The intuition would be using a pre-trained classifier to classify which generated image is like the ones used in the training data. By using this classifier, I can gain a better intuition about the performance of my GAN and which classes are lacking behind.

Conditional GAN ResNet32 Evaluation

	precision	recall	f1-score	support
Airplane	0.67	0.71	0.69	1000
Automobile	0.85	0.71	0.77	1000
Bird	0.58	0.45	0.51	1000
Cat	0.45	0.38	0.41	1000
Deer	0.53	0.66	0.59	1000
Dog	0.52	0.41	0.46	1000
Frog	0.65	0.81	0.72	1000
Horse	0.67	0.69	0.68	1000
Ship	0.72	0.83	0.77	1000
Truck	0.75	0.74	0.75	1000
accuracy			0.64	10000
macro avg	0.64	0.64	0.63	10000
weighted avg	0.64	0.64	0.63	10000

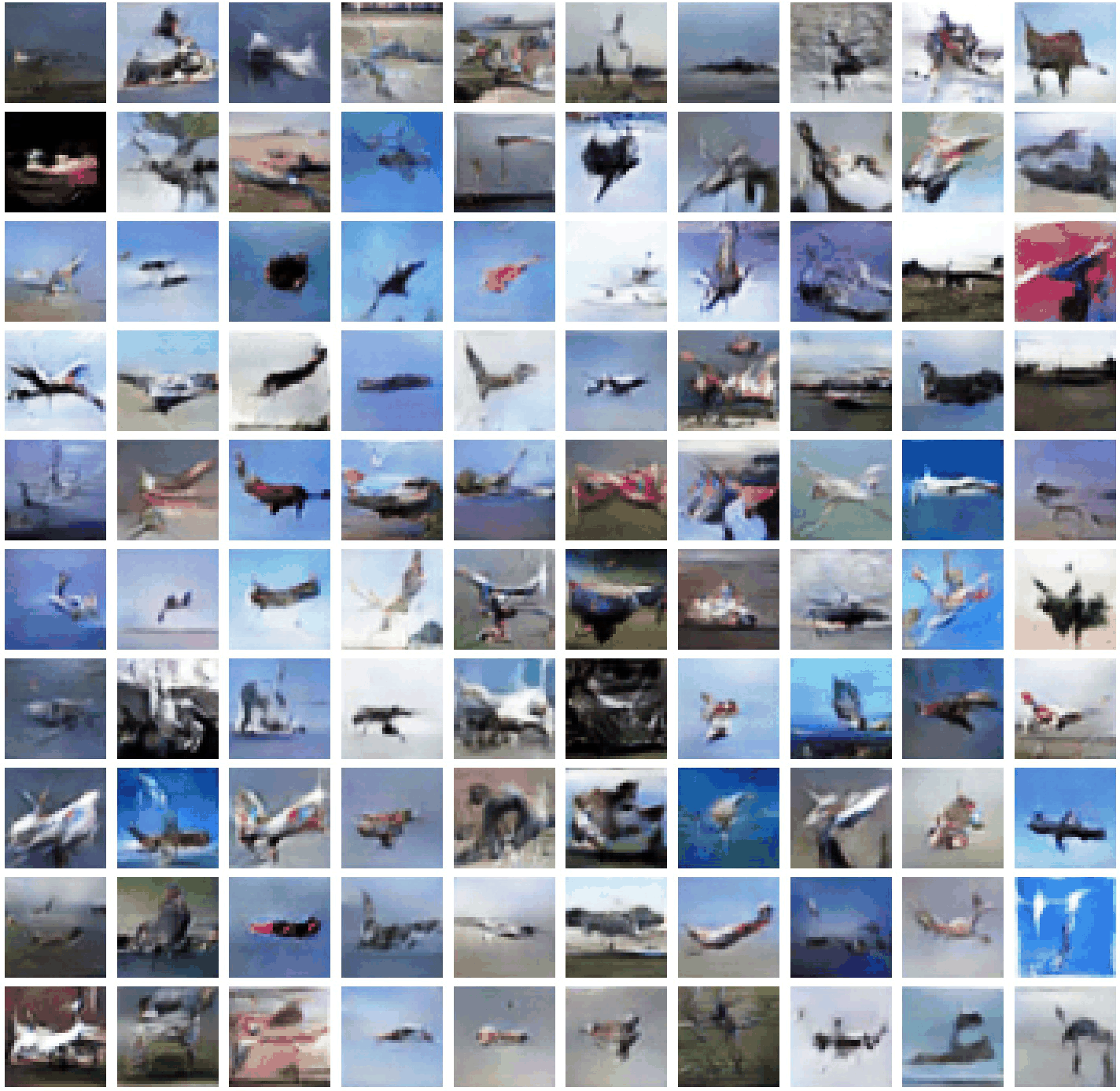
Conditional GAN ResNet32 Evaluation



Conditional GAN Quantitative Results

Architecture	Epoch Checkpoint	FID	KL Divergence	ResNet34 Accuracy
CGAN	100	90.918 +- 0.325	4.6261	0.64

airplane



Conditional GAN Conclusion

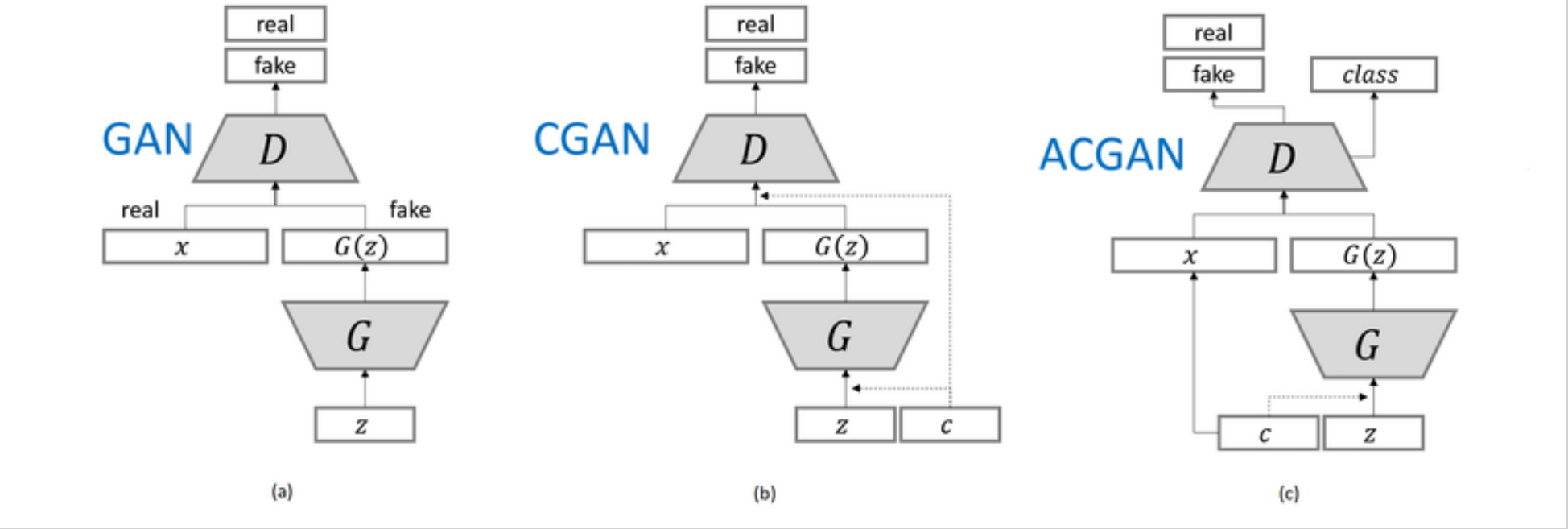
- Network was able to train until the pre-set 100 epochs without any issues.
- Fréchet Inception Distance (10,000 sample): 90.918 +- 0.325
- ResNet32 seems to predict only 64% of the synthetic images accurately.
- Majority of **Ship**, **Frogs**, and **Truck** are correctly identified by the image classifier; indicates Generator can generate images of these classes very well.
- Majority of Cat, Dog, and Bird are misclassified by the image classifier; indicates Generator is unable to generate images of these classes very well.
- Animal classes (e.g. Bird, Cat, Deer, Dog) has a tendency to have a false positive error, indicates Generator has difficulty producing images with animal class.

Conditional GAN
Further Improvements

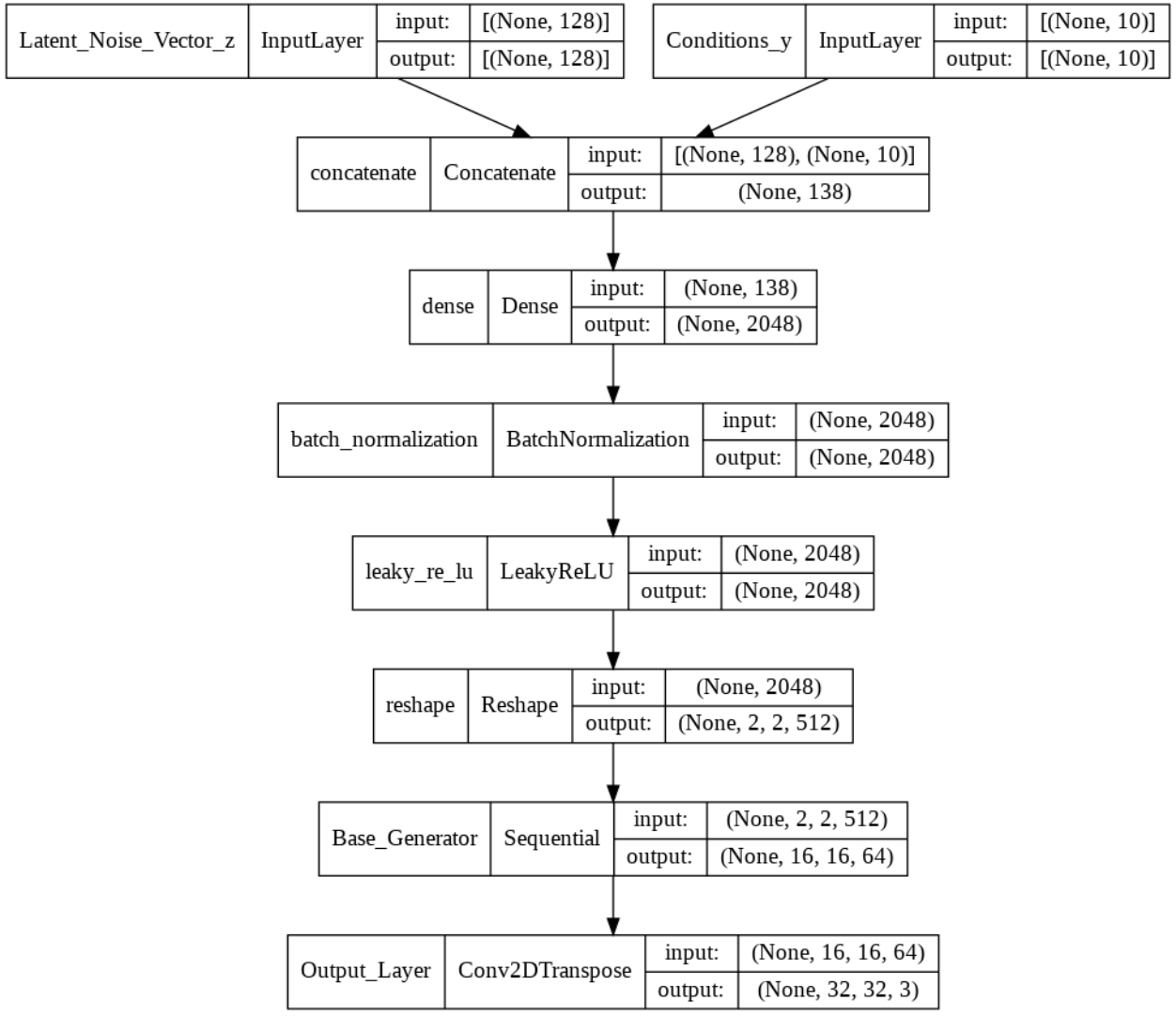
- Perhaps making better use of the Labels could improve image generation quality (ACGAN, Semi-Supervised GAN)
- Changing Loss Functions, e.g. Hinge Loss, Wasserstein Loss
- Label Smoothing
- Multiple Generator Update per Discriminator Update
- Differentiable Augmentation

Auxiliary Classifier GAN

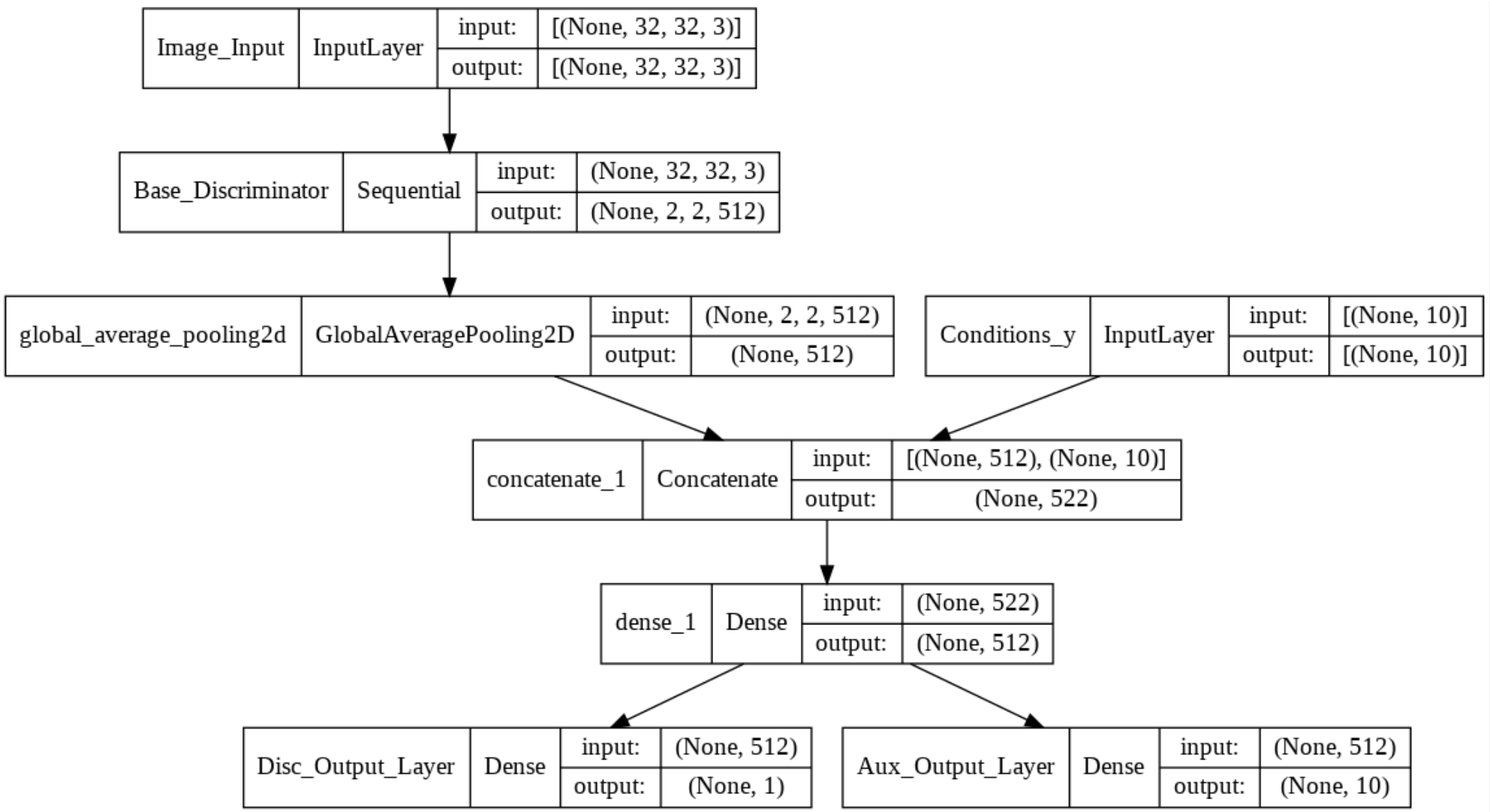
Auxiliary Classifier Generative Adversarial Network, or ACGAN in short, follows a similar architecture to a Conditional GAN. In fact, I've only added less than 10 lines of code for this implementation. There are two main difference between the two architectures - the discriminator produces two outputs, and an auxiliary loss function is included in the training process.



Auxiliary Classifier GAN Generator



Auxiliary Classifier GAN Discriminator



Auxiliary Classifier GAN Loss Function

There is an important concept for AC-GAN during training process. Instead of having one loss function for the discriminator, there are two loss function for different purpose.

- **Discriminator Loss**: Error between actual image and generated image (Binary Cross Entropy)
- **Auxiliary Classifier Loss**: Error between actual conditions and predicted conditions (Categorical Cross Entropy)

Total Loss = Discriminator Loss + Auxiliary Classifier Loss

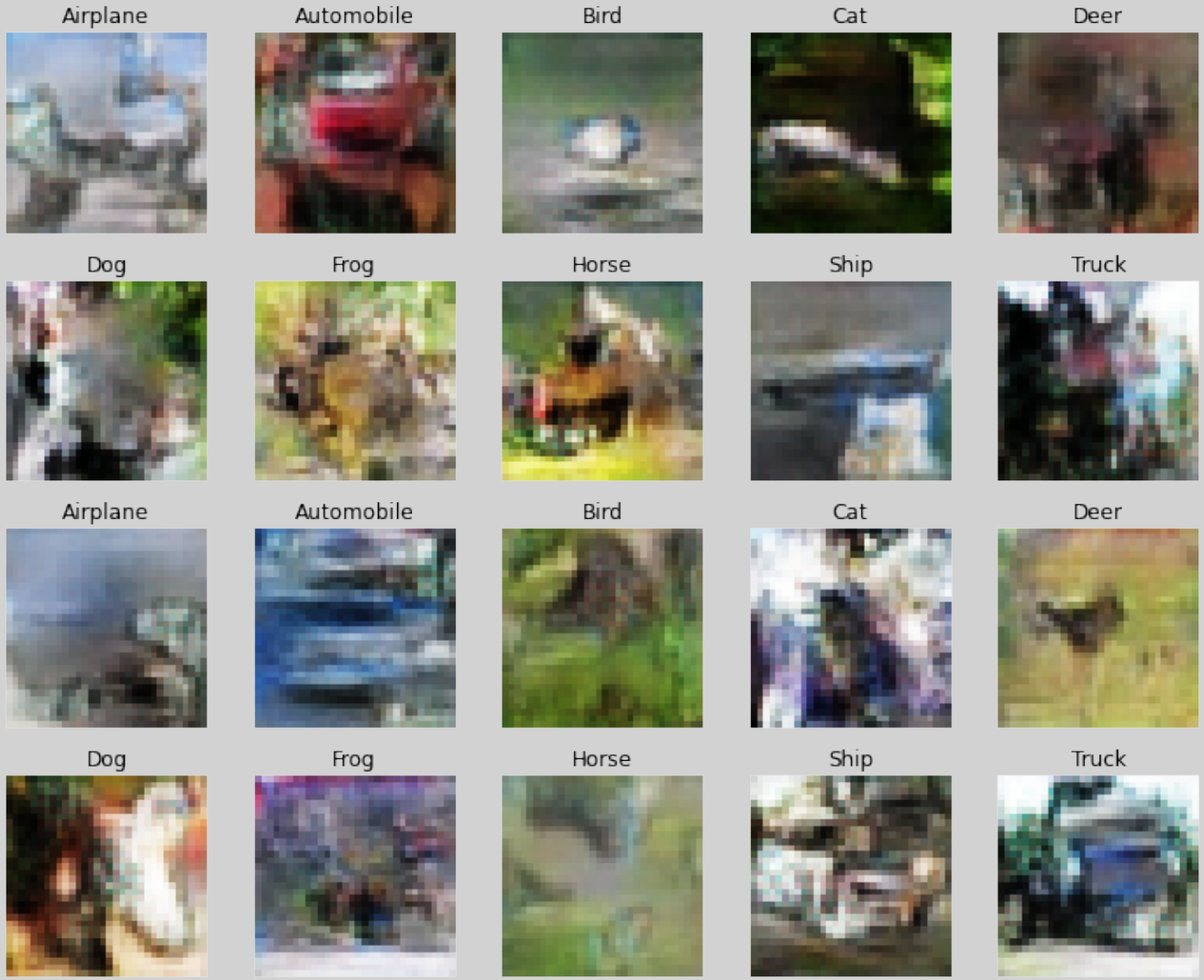
Auxiliary Classifier GAN Training Function

```
# 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)
)
```

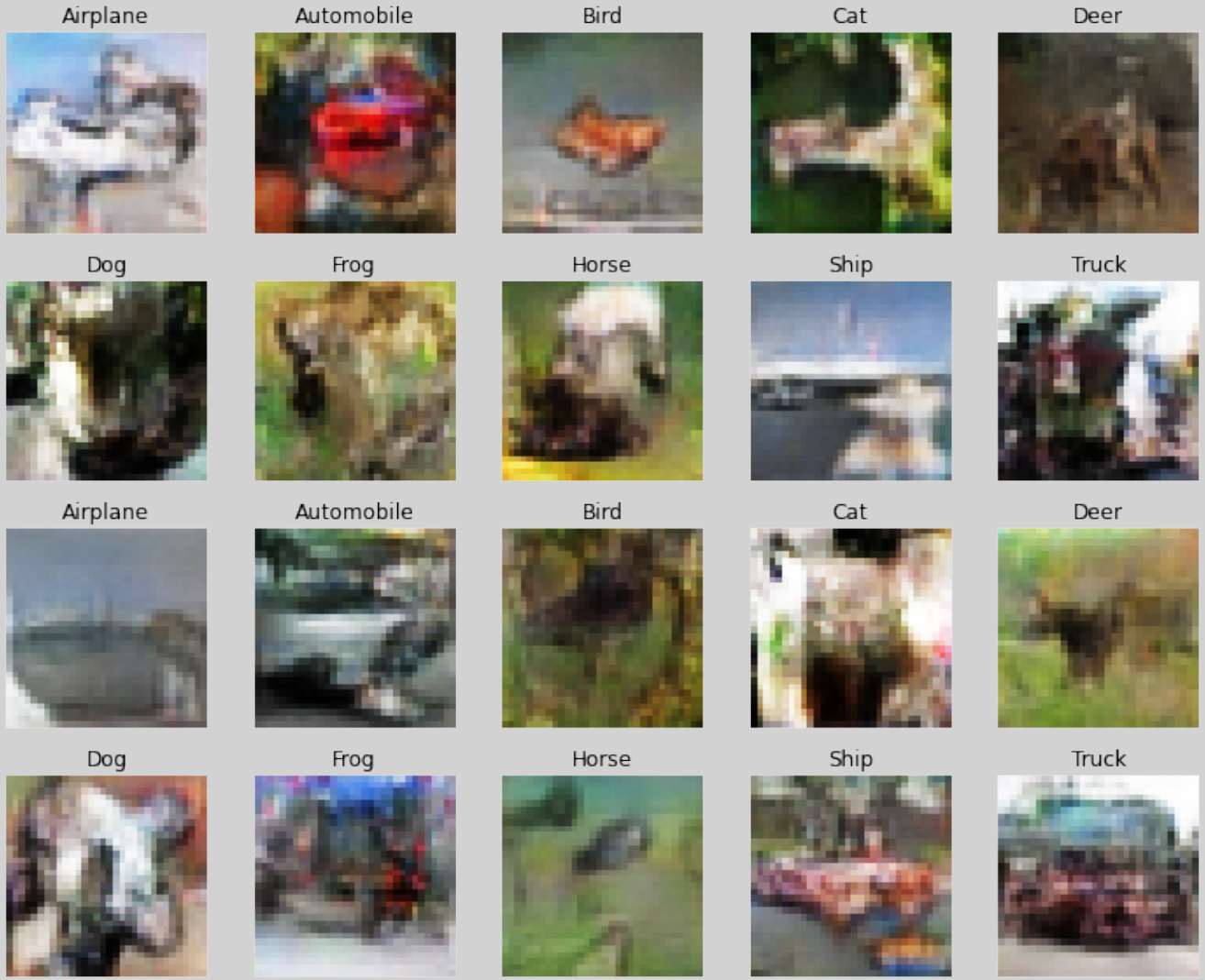
```
# Train the Generator
with tf.GradientTape() as tape:
    fake_images = self.generator([latent_noise_vector, condition])
    disc_output, aux_output = self.discriminator([fake_images, condition])
    disc_g_loss = self.disc_loss_fn(misleading_labels, disc_output)
    aux_g_loss = self.aux_loss_fn(condition, 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))
```

Auxiliary Classifier GAN
Epoch 0

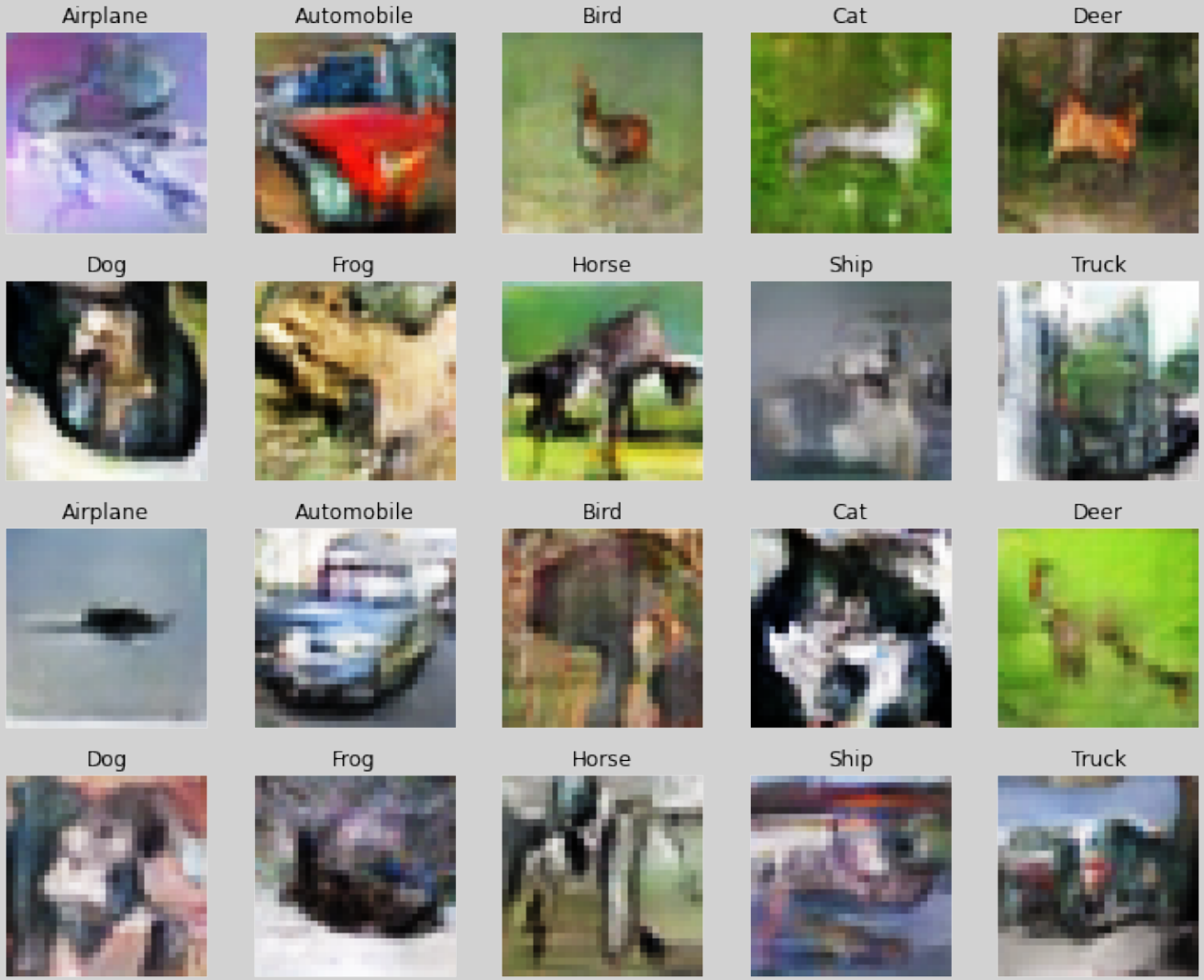
Auxiliary Classifier GAN
Epoch 20



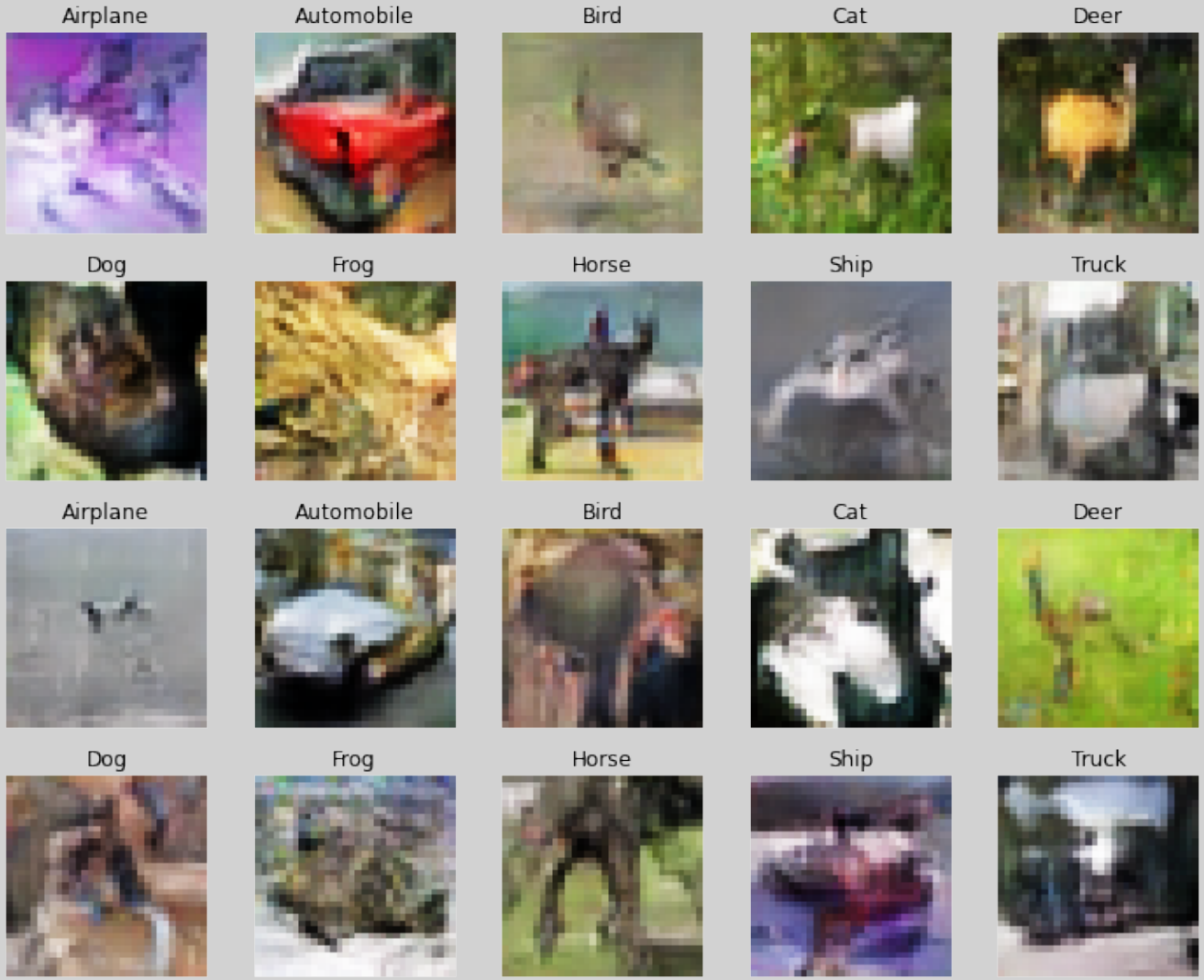
Auxiliary Classifier GAN
Epoch 40



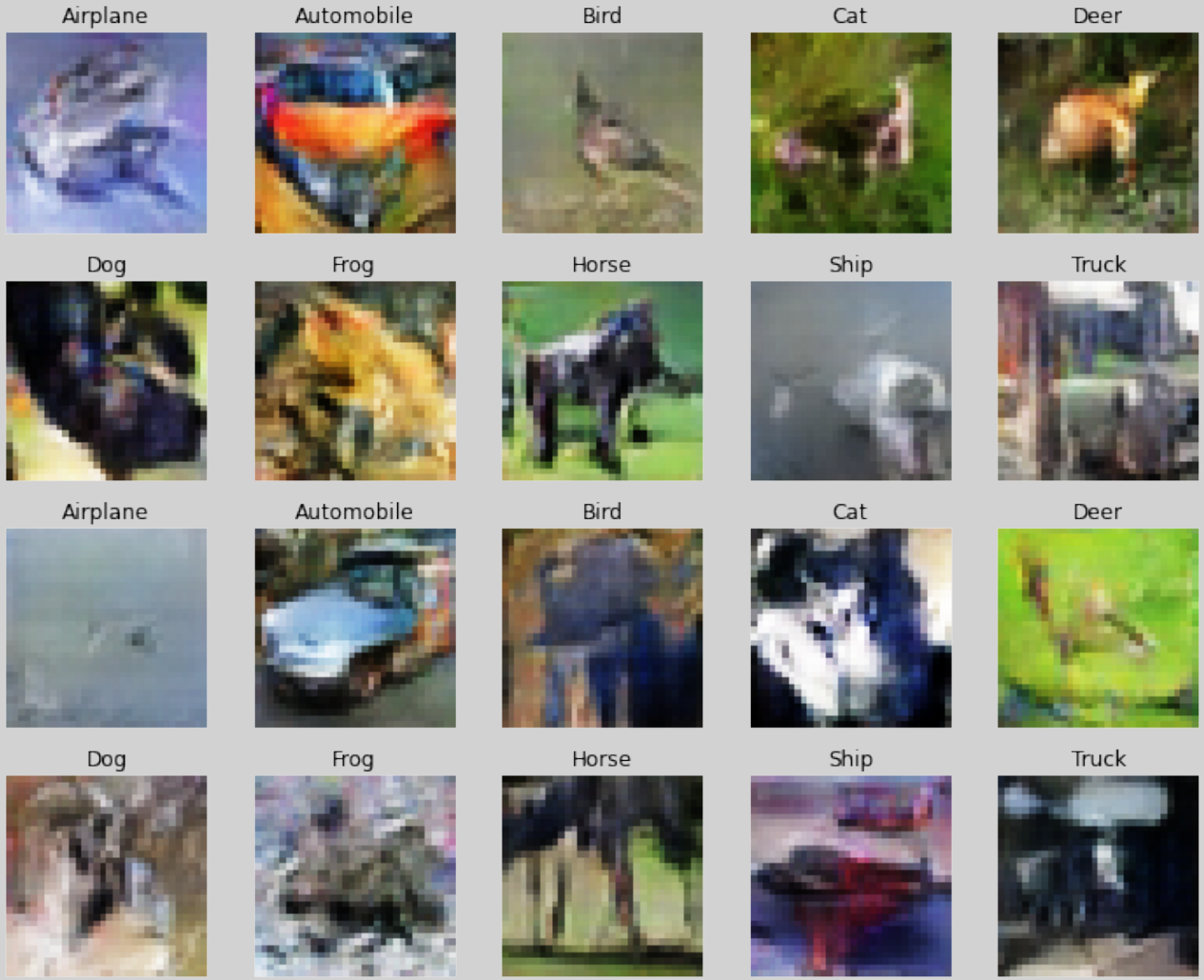
Auxiliary Classifier GAN
Epoch 60



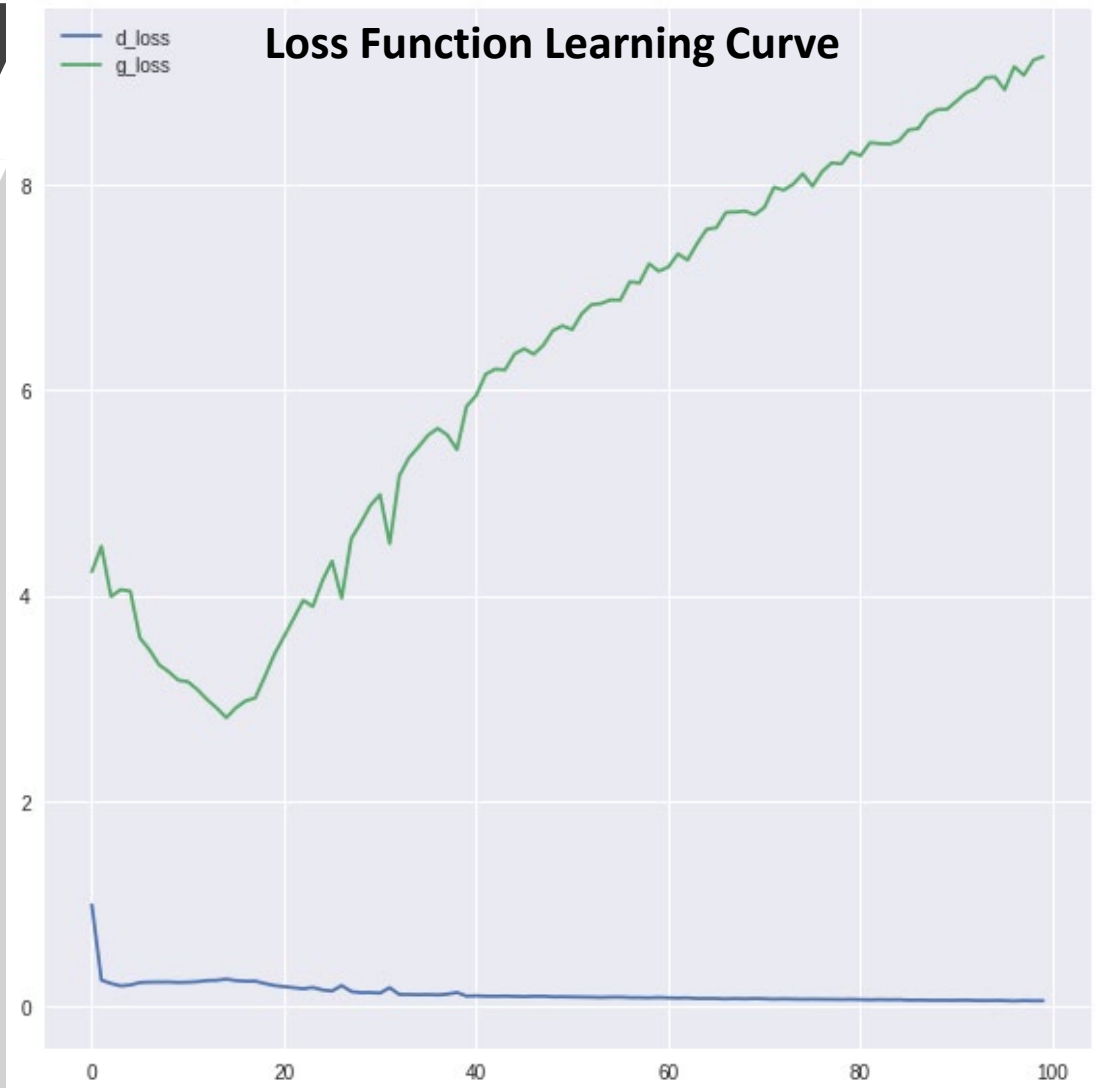
Auxiliary Classifier GAN
Epoch 80

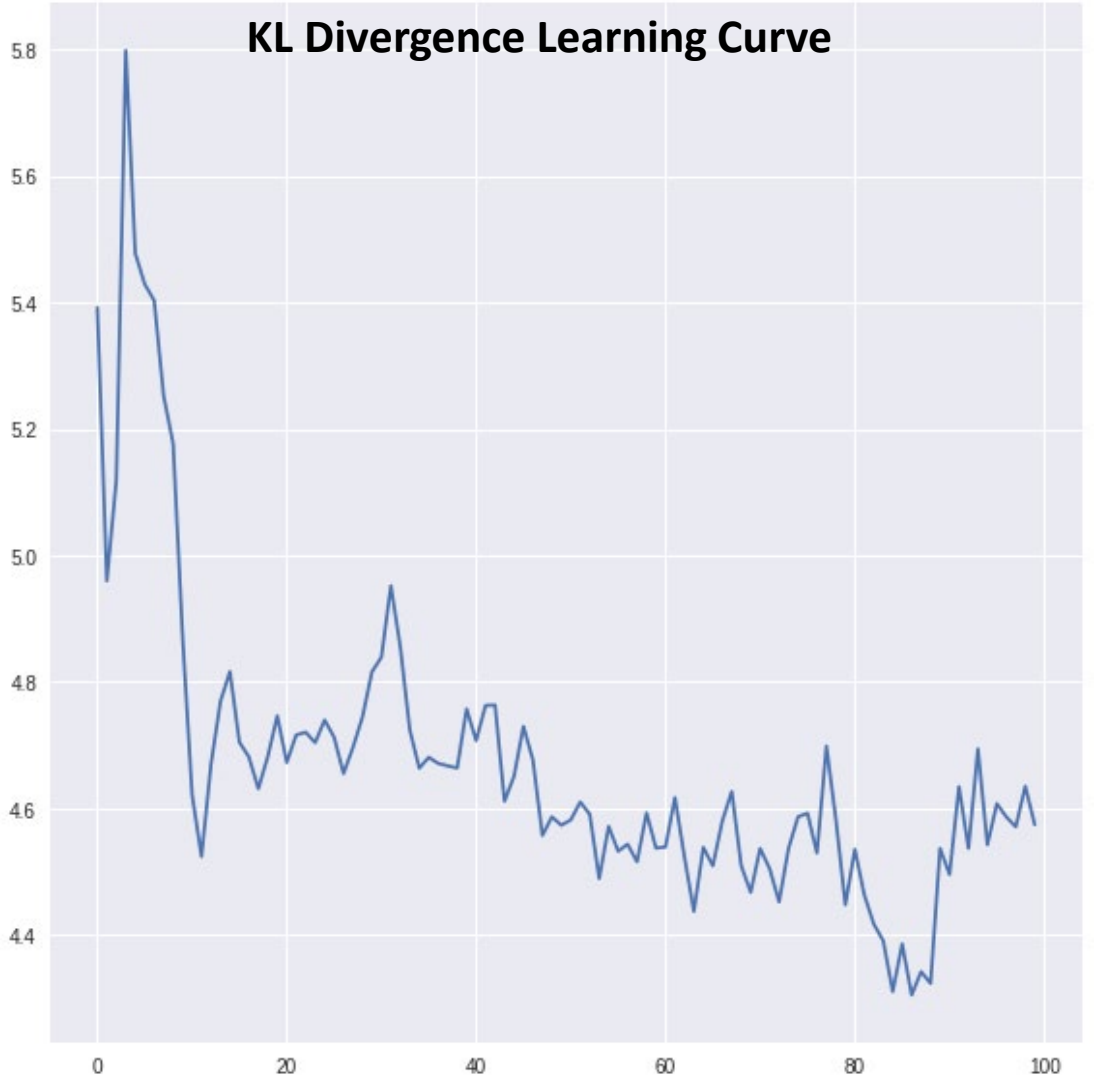


Auxiliary Classifier GAN
Epoch 100



Auxiliary Classifier GAN Learning Curve

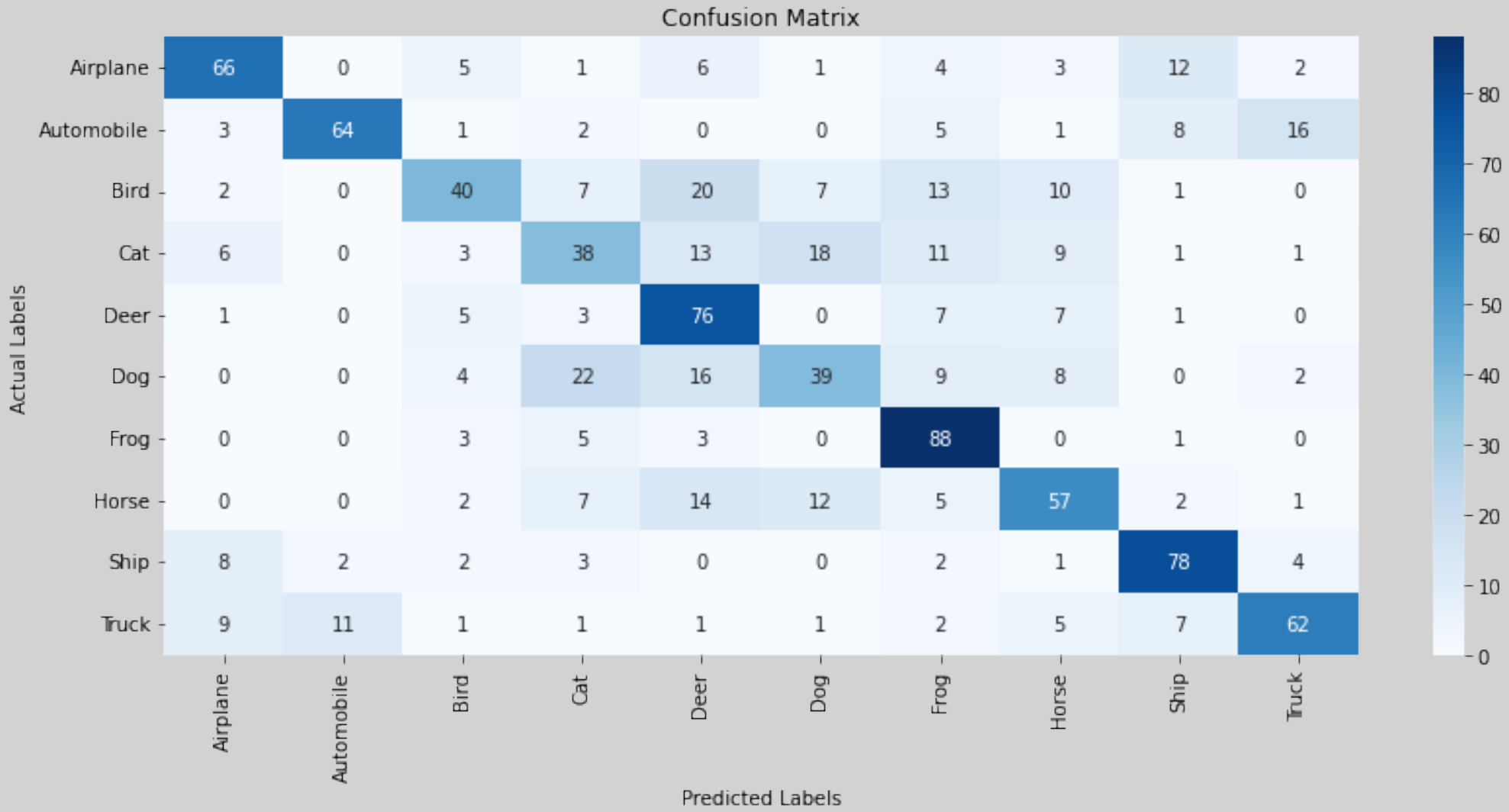




Auxiliary Classifier GAN ResNet32 Evaluation

	precision	recall	f1-score	support
Airplane	0.69	0.66	0.68	100
Automobile	0.83	0.64	0.72	100
Bird	0.61	0.40	0.48	100
Cat	0.43	0.38	0.40	100
Deer	0.51	0.76	0.61	100
Dog	0.50	0.39	0.44	100
Frog	0.60	0.88	0.72	100
Horse	0.56	0.57	0.57	100
Ship	0.70	0.78	0.74	100
Truck	0.70	0.62	0.66	100
accuracy			0.61	1000
macro avg	0.61	0.61	0.60	1000
weighted avg	0.61	0.61	0.60	1000

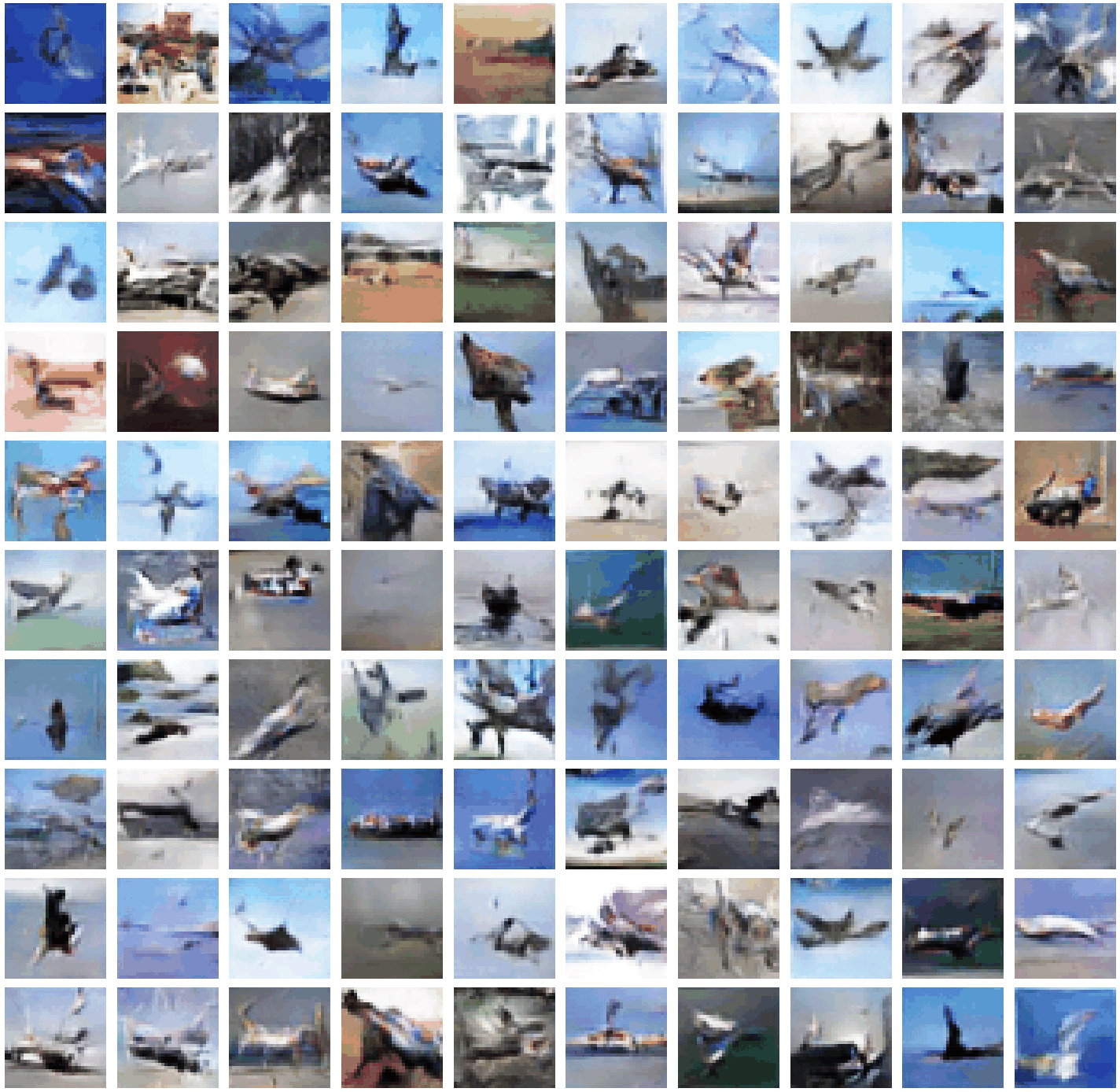
Auxiliary Classifier GAN ResNet32 Evaluation



Auxiliary Classifier GAN Quantitative Results

Architecture	Epoch Checkpoint	FID	KL Divergence	ResNet34 Accuracy
ACGAN	80	93.149 +- 0.737	4.4176	0.61

airplane



Auxiliary Classifier GAN Conclusion

- KL Divergence indicates that the model trains very well until epoch 80 - fail collapse problem.
- Fréchet Inception Distance (10,000 samples): 93.149 +- 0.737
- ResNet32 seems to predict only 61% of the synthetic images accurately.
- Majority of **Frogs**, **Ship**, and **Truck** are correctly identified by the image classifier; indicates Generator can generate images of these classes very well.
- Majority of **Bird**, **Dog**, and **Cat** are misclassified by the image classifier; indicates Generator is unable to generate images of these classes very well.
- Animal classes (e.g. Bird, Cat, Deer, Dog) tends to have a false positive error, indicates Generator has difficulty producing images with animal class.

Auxiliary Classifier GAN Further Improvement

- Attempt to change loss function, e.g. Hinge Loss, Wasserstein Loss
- Label Smoothing
- Multiple Generator Update per Discriminator Update
- Differentiable Augmentation

Final Quantitative Results

Architecture	Epoch Checkpoint	FID	KL Divergence	ResNet34 Accuracy
GAN	40	138.33 +- 0.837	5.6873	-
CGAN	100	90.918 +- 0.325	4.6261	0.64
ACGAN	80	93.149 +- 0.737	4.4176	0.61

Conclusion

- I have attempted three different GAN architectures – DCGAN, CGAN, ACGAN
- Based on these implementation, I have learnt that every little details in the architecture does matter to training.
- Probably is the reason why there is so many research conducted on GAN.

- Further Improvements that could not have been implemented due to time constraint – changing loss function, label smoothing, multiple time update rule, differentiable augmentation, skip connections, and many more.